▶ A scuola con PC Open

Web Developer PHP

di Federico Pozzato

1 La scelta di PHP

Citando testualmente la prefazione del manuale ufficiale di PHP.net: "L'obiettivo principale del linguaggio PHP è di permettere agli sviluppatori Web di scrivere velocemente pagine Web dinamiche, ma con PHP si possono fare molte altre cose".

È in questo spirito che *PC Open* introduce nelle sue pagine questo minicorso di PHP, potente linguaggio di scripting open source, pienamente integrabile con HTML (da cui la propedeuticità del corso Webmaster, fornito sul *CD Guida n. 2* unito a questo numero, utile anche se non necessaria) e indirizzato prevalentemente allo sviluppo dei siti Internet dinamici. PHP viene utilizzato anche per creare scripting di righe comando e applicazioni client-side **GUI** (Graphical User Interface) utilizzando le estensioni PHP-GTK. Tralasciando queste due applicazioni specialistiche, vedremo come PHP offra strumenti professionali molto evoluti per la gestione Web, pur mantenendo una struttura semplice, adatta anche a chi si avvicina per la prima volta a linguaggi di programmazione di questo tipo.

Alla fine di questo minicorso avremo gli strumenti di base per migliorare i nostri siti e per affrontare una delle più importanti funzionalità di PHP, ossia l'integrazione con i database e in particolare con MySQL.

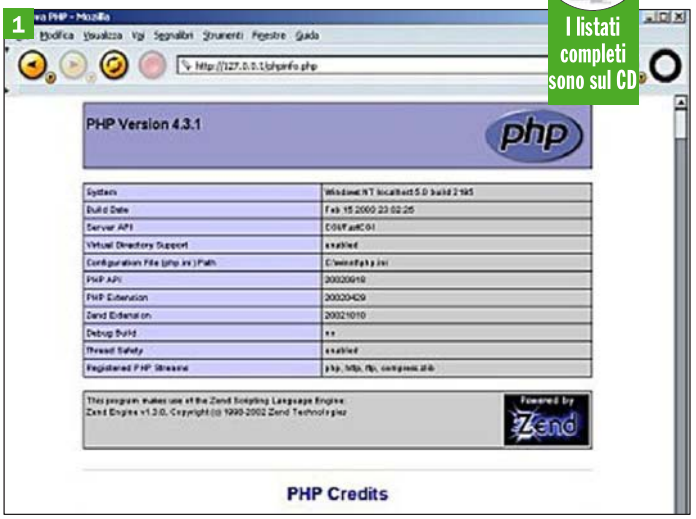
La storia del linguaggio PHP

PHP è un acronimo ricorsivo (tipico dell'ambiente open source) che significa *PHP: Hypertext Preprocessor*. Che cosa sia un ipertesto è chiaro, mentre può lasciare perplessi il significato di Preprocessor: PHP, come **ASP**, è un linguaggio **server-side**, ossia il codice PHP è prima elaborato dal server e solo dopo indirizzato al browser che ha chiamato la pagina. In tal senso PHP elabora la pagina "prima" della sua visualizzazione.

È una caratteristica totalmente diversa rispetto a linguaggi come Javascript che invece sono **client-side**, ossia interamente interpretati dai browser. Vedremo in seguito come questa caratteristica server-side consenta l'implementazione di strutture e funzionalità assolutamente inedite per chi è abituato a ragionare in termini di HTML puro.

PHP consente, inoltre, di creare immagini, file PDF, filmati Flash, generare file basati su XML, utilizzare i protocolli di posta POP3 e IMAP (ne vedremo un'applicazione nella seconda parte del corso), comprimere file in gzip e bz2, gestire sessioni, utilizzare funzioni e classi, fare uso della programmazione orientata agli oggetti e naturalmente interagire con tutti i database più diffusi. Per alcune di queste funzioni bisogna installare estensioni fornite con il file binario di PHP e quindi si dovrà verificare presso il proprio Web provider l'installazione delle specifiche librerie (ad esempio quelle grafiche).

PHP ha ormai quasi 10 anni di storia alle spalle (è stato creato nel 1994 da Rasmus Lerdorf e la prima distribuzione risale al 1995; sito di riferimento: www.php.net), sposa appieno la filosofia open source ed è distribuito con licenza GPL. È un **linguaggio multipiattaforma**, utilizzabile indifferentemente su macchine client con sistemi operativi Windows, Linux, BSD o Mac e interfacciabile con tutti i più popolari Web server.



La pagina phpinfo.php *è la prima pagina di questo corso realizzata in PHP*

Utilizzare PHP off line e scrivere i primi listati

I listati in PHP possono essere creati utilizzando un qualsiasi editor di testo e salvando le pagine con estensione *.php*. Proviamo a scrivere il nostro primo semplice listato (la parte PHP è in grassetto. Più avanti ne approfondiremo il significato), salviamolo col nome *phpinfo.php* (*listato1*) e poi visualizziamolo col nostro browser:

LISTATO 1 (phpinfo.php)

```
<html>
  <head>
    <title>Prova PHP</title>
  </head>
  <body>
    <?php
      phpinfo();
    ?>
  </body>
</html>
```

IL CALENDARIO DELLE LEZIONI

▶ Lezione 1:
- **La scelta di PHP**
- **PHP con un server off line**
- **La prima pagina PHP:** inseriamo PHP in HTML
- **Funzioni base e variabili**
- **I costrutti di controllo:** if else, while, do until, foreach
- **I form:** passaggio di dati coi metodi GET e POST
- **Esempi**

Le prossime puntate

Lezione 2: Approfondiamo PHP
Lezione 3: PHP e i database
Lezione 4: PHP e MySql
Lezione 5: Gestire un sito dinamico con PHP e MySql

Sul vostro browser vedrete o una pagina totalmente vuota oppure il listato esattamente come è scritto qui sopra. Cos'è successo? Abbiamo affermato che PHP è un linguaggio server-side, quindi per visualizzare le pagine create abbiamo bisogno di tre strumenti: un server Web, il supporto PHP (il **parser**, gli elementi di una frase o di un'istruzione) attivato da parte del server scelto e un browser.

Nel nostro esempio ci siamo serviti solamente del browser che, da solo, non è in grado di interpretare la pagina PHP e la fa vedere come fosse un normale documento HTML (quindi, in questo caso, apparentemente vuoto) oppure come un file testo. Nel primo caso la pagina sembra vuota, ma se andiamo a vedere il codice sorgente (ogni browser consente di farlo selezionando l'apposita voce presente nei menu) vedremo il *listato 1*. Per vedere invece la pagina PHP interpretata nella maniera esatta, avremmo dovuto per prima cosa caricare la pagina *phpinfo.php* su un server Web (abilitato per la traduzione delle pagine PHP) e poi richiamare la pagina col browser (vedi *disegno a fondo pagina*). Avremmo ottenuto quanto visibile *nell'immagine 1*: la nostra prima pagina ci restituisce tutte le caratteristiche della versione PHP installata sul server Web.

Per la maggior parte delle persone è certamente improponibile pensare di poter imparare a programmare in PHP restando connessi a Internet, vuoi a causa dei costi, vuoi per la scomodità di dover ogni volta caricare le proprie pagine via FTP (o con upload dedicati) per poi richiamarle tramite browser (vedi *disegno A*).

Risulta quindi necessario installare un server sul proprio PC, col supporto alle pagine PHP per le operazioni di parsing. L'installazione di un server sul proprio PC è già stata descritta nella 7ª lezione del corso Webmaster e la si dà per acquisita (l'installazione del server Web va fatta prima di PHP). Vediamo adesso come installare il parser, con indicazioni riguardanti tre server: Apache, Xitami e IIS di Microsoft.

La prima regola fondamentale è scaricare l'ultima release stabile di PHP per il proprio sistema direttamente dal sito ufficiale: www.php.net.

Al momento in cui scriviamo, la release stabile è una 4.3.x, in attesa del sospirato rilascio della release 5. Tralasciamo l'installazione di PHP su Linux, in quanto è un'operazione generalmente eseguita in automatico scegliendo gli opportuni pacchetti della propria distribuzione, e dedichiamoci all'ambiente Windows. Nell'area download di www.php.net si trovano due file utilizzabili con sistemi Windows:

- un file installer (come: *php-4.3.7-installer.exe*). Questo file ha dimensioni minime (circa 1 MB), è direttamente eseguibile nel sistema e configura automaticamente i server IIS e Xitami, consentendo l'installazione come versione CGI per il server Apache. L'installer non comprende le estensioni di PHP.
- un file zippato contenente il codice binario (come: *php-4.3.7-Win32.zip*). Questo file ha dimensioni ben più consistenti (circa 7 MB), ma consente l'installazione (manuale) di tutti i moduli.

Il server Apache

Come ASP richiamava il server IIS, così PHP richiama il server Apache. Tuttavia, non esiste al momento una procedura automatica per configurare il server Apache, sia esso 1.3.x o 2.0.x (supporto pienamente abilitato solo con una versione di PHP oltre la 4.3.1).

Si può procedere in due modi, configurando PHP in versione CGI (installazione più semplice) o come modulo (installazione consigliata per motivi di sicurezza).

Nel primo caso utilizziamo il file *Installer*. Eseguiamolo (ricordandoci la cartella dove PHP verrà installato: *C:\PHP* va benissimo, ma potete cambiarla a piacere) e arriviamo al termine dell'installazione (anche scegliendo l'opzione di configurazione automatica di Apache, il software vi risponderà che questa procedura non è ancora applicabile). Ora dovete editare il file *httpd.conf* del server Apache (*Start* > *Programmi* > *Apache* > *Configure* > *Edit the httpd.conf configuration file*) e, dopo esservi posizionati nella sezione *ScriptAlias*, aggiungete le seguenti righe:

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php
"/php/php.exe"
```

sostituendo a *c:\php* la cartella di installazione. PHP viene installato nella versione CGI: fate ripartire il server Apache e poi verificate se la pagina *phpinfo.php* viene visualizzata nella maniera corretta.

Questo tipo di installazione è giudicato non sicuro e meno performante dell'installazione di PHP come modulo, però dal momento che ce ne serviamo solo per scopi didattici (esclusivamente off line) va comunque bene per i nostri scopi. Non è possibile, però, installare alcuna estensione.

Il secondo metodo richiede il file zip binario. Per prima cosa si deve scompattare il file in una cartella tipo *c:\php*, quindi copiare il file *php.ini-dist* nella *%SYSTEMROOT%* (per Windows 2000 è *C:\WINNT*) modificandone il nome in *php.ini*. Altra operazione da compiere è copiare il file *php4ts.dll* nella cartella *c:\php\sapi*. Ultimi sforzi: editare il file *httpd.conf* del server Apache (*Start* > *Programmi* > *Apache* > *Configure* > *Edit the httpd.conf configuration file*) e, dopo essersi posizionati nella sezione *LoadModule*, aggiungere le seguenti righe (per Apache 2.0.x):

```
LoadModule php4_module
c:/php/sapi/php4apache2.dll  [o
php4apache.dll per Apache 1.3.x]
AddType application/x-httpd-php .php
```

Facciamo le necessarie verifiche, dopodiché siamo pronti per iniziare.

IIS, PWS e Xitami

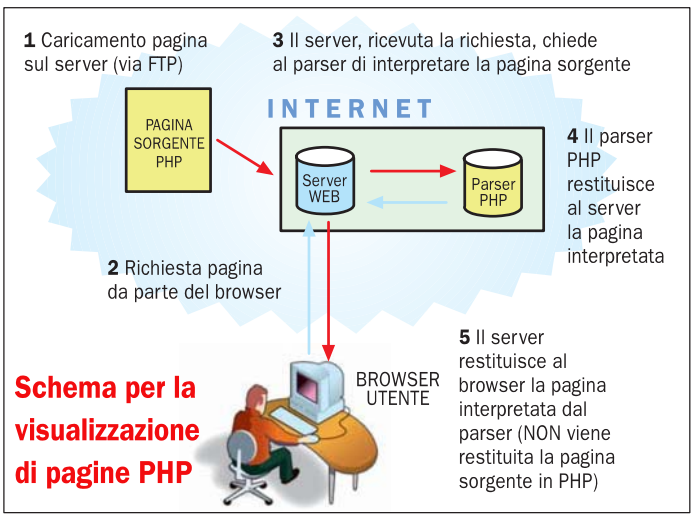
Con IIS, PWS e Xitami la vita è più semplice rispetto ad Apache, sempre che ci si accontenti dell'installazione minima (generalmente sufficiente per i nostri scopi).

In questo caso si deve solo eseguire l'installer PHP: durante la procedura di installazione ci verrà chiesto se è presente un server da configurare con PHP. Scegliete IIS o Xitami e attendete la conclusione dell'install shield: da adesso saremo in grado di verificare off line le nostre pagine PHP. Per verificare il funzionamento, fate partire il server, provate a inserire nella cartella scelta come root dei documenti il file *phpinfo.php* e richiamatelo col browser: dovete ottenere l'*immagine 1* della pagina a lato.

L'installazione delle estensioni implica l'utilizzo del file zip binario e qualche passo "manuale" aggiuntivo. Non la trattiamo in questa sede, ma è comunque presente il file *install.txt* che vi assiste passo passo.

Estensioni

Per utilizzare le estensioni si deve effettuare l'installazione del parser PHP tramite il file zip binario. Andrà poi modificato il file *php.ini* (dalla *%SYSTEMROOT%*) inserendo l'indicazione *C:\php\extensions* dopo l'uguale nella riga che inizia con *extension_dir =*. Fatto questo, sempre nel file *php.ini*, cercare la sezione *Windows extensions* e togliere il punto e virgola davanti alle estensioni da abilitare (ad esempio, per utilizzare le funzioni PDF bisogna abilitare l'estensione *php_pdf.dll*).



Schema per la visualizzazione di pagine PHP

2 La prima pagina in PHP

La prima pagina PHP, in realtà, l'abbiamo già realizzata (la solita *phpinfo.php*) e la possiamo quindi analizzare. La parte iniziale e la parte finale comprendono solo tag e contenuti HTML, mentre la "sezione" PHP, quella interpretata dal parser tramite il server e restituita poi al client browser, viene chiaramente identificata dai tag *<?php* (inizio sezione PHP) e *?>* (fine sezione PHP).

Questi tag possono essere aperti e chiusi quante volte si vuole in un listato HTML: ai fini dell'efficienza nell'elaborazione della pagina, infatti, è meglio uscire dalla modalità PHP se si devono passare blocchi di contenuti HTML "puri" (vale specialmente per pagine complicate).

Tutto quello che si trova compreso tra i tag PHP viene quindi interpretato dal server e restituito al browser per la visualizzazione: la funzione *phpinfo()*, infatti, restituisce tutte le caratteristiche settate per il parser PHP installato nel server. Può essere curioso esaminare il sorgente di questa pagina: sono tutti contenuti HTML e della funzione *phpinfo()*, giustamente, non c'è alcuna traccia.

Notate il ";" al termine della riga compresa nei tag: tutte le istruzioni PHP terminano con questo segnale di fine riga. Il tag di chiusura *?>* fa anche da ";" e quindi per l'ultima istruzione prima della chiusura si potrebbe anche eliminare il ";", però è certamente meglio non prendere certe abitudini e usare tutta la sintassi normalmente richiesta.

Un secondo esempio, *hello.php*, ci consente di introdurre la funzione *echo* che ha lo scopo di restituire un output (*listato 2 - hello.php*):

LISTATO 2 (hello.php)
```
<?php
  echo "ciao PC OPEN";
?>
```

La funzione *print()* produce esattamente lo stesso output.

È possibile anche introdurre dei tag HTML all'interno della sezione PHP come si può vedere da *hello-html.php* (*listato 3*). Andranno inseriti all'interno di *echo* in quanto questi tag HTML devono diventare un output del parser:

LISTATO 3
```
<?php
    echo "<h1>Ciao PC OPEN</h1>";
?>
```

Per indicare inizio e fine della sezione PHP

Ci sono altri sistemi per indicare l'inizio e la fine della sezione PHP:

- **tag brevi** (short tag): basta indicare *<?* e *?>*. Per usarli deve essere appositamente configurato il parser PHP del server Web ospitante. Generalmente è così, ma l'uso di questi short tag è comunque sconsigliato per motivi di sicurezza: se, per qualunque motivo (manutenzione, aggiornamento, e così via), il parser non interpretasse più in maniera corretta gli short tag, il vostro codice diverrebbe visibile a tutti (come nell'esempio visto all'inizio).
- **indicazione di script:** *<script language="php">* e *</script>*. Non è sempre interpretato correttamente quindi evitiamolo.
- **ASP tag**: i tag sono gli stessi usati per ASP: *<%* e *%>*. Anche in questo caso deve essere attivata un'apposita opzione del parser (*asp_tags = On*), con gli stessi problemi di sicurezza già indicati per gli short tag.

In *tag.php* trovate riuniti insieme questi ultimi tre modi di indicare il codice PHP (vedi *listato 4*).

Per buona regola, è sempre bene utilizzare *<?php* e cercare di essere il più ordinati possibile, nell'eventualità di dover fare delle modifiche a distanza di tempo.

Per inserire dei commenti si premettono al commento le doppie slash *//*.

Curiosità

In un'ottica di filosofia open source si sente nominare l'acronimo **LAMP**. Esso indica un gruppo di prodotti divenuti, nel loro insieme, un riferimento per lo sviluppo di siti Web:

Linux
Apache
MySQL
PHP

3 Tipi di dato, variabili e operatori

Per iniziare realmente a usare PHP e scoprirne le caratteristiche, abbiamo bisogno di spendere ancora qualche minuto per introdurre la gestione delle variabili.

Diversamente da ASP, con PHP non serve dichiarare esplicitamente le variabili indicando nome e tipo, ma, più semplicemente, basta assegnare alla variabile il suo valore: sarà il parser ad attribuire alla variabile il giusto "tipo" (dichiarazione implicita). Tutte le variabili vanno indicate col simbolo *$* davanti al nome scelto (il primo digit dopo *$* non deve mai essere un numero).

Tipi di variabili scalari

Vediamo i quattro tipi di variabile scalare che abbiamo a disposizione:

```
$prova1 = 12;
// $prova1 è un tipo di dato intero (integer),
```

in questo esempio è un intero a base decimale, ma avremmo potuto fare una dichiarazione con le notazioni esadecimale o ottale. In caso di superamento del limite massimo previsto per un intero, PHP assegna automaticamente alla variabile il tipo float.

```
$prova2 = 1,23;
// $prova2 è un tipo di dato
a virgola mobile (float).

$prova3 = TRUE;
// $prova3 è un tipo di dato
booleano (boolean),
```

ossia assume solo valori vero (true) o falso (false). Il numero 0 o una stringa vuota corrispondono al valore booleno *false*; un numero diverso da 0 o una stringa non vuota corrispondono al valore booleano *true*.

```
$prova4 = "pippo";
// $prova4 è un tipo di dato
stringa (string).
```

La stringa può essere definita utilizzando le virgolette singole ' (*single quotes*) o le virgolette doppie " (*double quotes*). Nel proseguimento del corso si userà la seconda notazione che permette, tra le altre cose, di inserire dei nomi di variabili all'interno della dichiarazione

```
$1prova = 1;
// il nome della variabile non
è valido (numero dopo $)
```

Ridichiarando le variabili, o concatenandole, PHP sceglie sempre automaticamente il tipo di dato da gestire. Se volessimo sapere, ad esempio per motivi di debug, quale sia il tipo di dato di una certa variabile, si può usare la funzione **gettype**:

```
<?php
  $prova = 1,2;
  echo gettype ($prova); // si ottiene a
  video il tipo di dato: float
?>
```

Il tipo può essere imposto mediante l'istruzione **settype** all'atto già della dichiarazione

(esplicita) della variabile o attraverso l'imposizione (**casting**) del tipo (boolean, int, float, string, array, object, null):

```
<?php
 setype ($prova, "int");
 $prova = 1,2;
 echo $prova; // si ottiene a video
 il valore del tipo "int": 1
?>

<?php
 $prova = 1,2;
 $intero = (int) $prova;
 echo $intero; // si ottiene a video
 il valore del tipo "int": 1
?>
```

Variabili di tipo composto

Oltre alle variabili di tipo scalare, ci sono due variabili di tipo composto: array e object.

Array non è solo il classico vettore (ossia una lista di elementi individuati da un indice numerico avente numero iniziale 0), ma può essere anche una tabella di hash (una mappa), dove un elemento è individuato da una chiave non necessariamente numerica. Ciò lascia estrema libertà di utilizzo del tipo array, ma impone anche di prestare attenzione quando viene richiamato un valore. Le chiavi possono essere valori sia numerici, sia di tipo stringa; gli elementi dell'array possono essere di qualunque tipo (anche altri array).

Se vogliamo creare un vettore, lo possiamo fare in maniera esplicita (indico la parola chiave *array*) o implicita (assegno i valori alla variabile che sarà il vettore):

```
$vettore_a = array (2, 5, 7, 9, 11);
// dichiarazione esplicita con elementi
solo numerici
 $vettore_b = array (2, "pippo", true,
1.24, array (1,2,3)); // dichiarazione
esplicita con elementi misti
 $vettore_c = array (0 => 2, 1 =>
"pippo", 2 => true); // notazione per
indicare le chiavi del vettore (potevano
essere omesse senza problemi)
 $vettore_d[0] = 2; // dichiarazione
implicita del primo elemento del
vettore_d
 $vettore_d [1] = "pippo";
// dichiarazione implicita del secondo
elemento del vettore_d
```

Per visualizzare un array e le sue chiavi usiamo la funzione **print_r**, come indicato nell'esempio *vettore.php* (*listato 5*). Dagli esempi qui sopra otteniamo coppie che sono sempre identificate da un numero. Per fare riferimento a un elemento specifico (per stamparlo, modificarlo e così via) si usa il costrutto:

```
 $array[n]
 con n valore intero che parte da 0.
```

Creare un array come tabella di **hash** è altrettanto semplice e versatile, ma in questo caso ci dovremo preoccupare di indicare anche il valore (sia esso un numero o una stringa) da assegnare alla chiave dell'elemento dell'array:

```
$hash_a = array ("italia" => "roma",
     "francia" => "parigi", "spagna" =>
     "madrid", 12 => "pippo");
$hash_b["italia"] = "roma";
$hash_b ["francia"] = "parigi";
$hash_b[12] = "pippo";
```

La stampa dell'hash la possiamo vedere nell'esempio *hash.php* (*listato_06*). Per eliminare, modificare, stampare i dati dell'hash vi faremo riferimento con i due costrutti (diversi a seconda del tipo di chiave):

```
 $array[numero]
 $array["chiave"] // o: $array['chiave']
```

Si rivelerà utile questa possibilità che ci concede PHP: dopo aver dichiarato un array *hash_a*, possiamo inserire gli elementi anche senza fare riferimento a una chiave specifica:

```
 $hash_a[ ] = "topolino";
```

In questo caso la chiave dell'elemento sarà il numero intero successivo al numero intero più alto già presente nell'hash. Pertanto, come mostrato dall'array *hash_b* dell'esempio *hash-due.php* (*listato 7*), possiamo creare facilmente un array assegnando come chiave iniziale un numero diverso da 0. Vedremo come questo possa essere utile quando parleremo di date.

Il tipo di dato **object** lo analizzeremo nel prosieguo del corso parlando delle classi.

Tipi di variabili speciali

Per finire l'analisi, vi sono ancora due tipi speciali di variabili: **resource** e **null**. Il primo è creato e usato da una serie di funzioni ben definite (per approfondimenti vedere le appendici del manuale PHP), mentre il secondo serve ad assegnare a una variabile lo speciale valore *NULL*.

Nell'ambito delle variabili, oltre a quelle definite da noi troviamo le cosiddette **variabili superglobals**, arrays contenenti informazioni sul Web server, sull'ambiente di utilizzo e sugli input degli utenti. Alcuni esempi si possono vedere (coi relativi valori) aprendo la nostra prima pagina creata (*phpinfo.php*), e altri li vedremo introducendo i form e le sessioni. Quasi tutti questi array superglobali iniziano con un underscore: *$_POST, $_GET, $_SESSION, S_SERVER*.

Operatori

Gli operatori consentono di eseguire operazioni sulle variabili, operazioni non solo di natura numerica (esempio: calcoli aritmetici) e testuale (esempio: concatenazione), ma anche operazioni di controllo (esempio: logica binaria) e confronto (esempio: confronto di valori e tipi).

La concatenazione è senza dubbio una delle operazioni più usate: le variabili in PHP possono essere concatenate usando il segno ".":

```
<?php
 $a = "PC OPEN";
 $b = "una rivista di informatica";
 $c = $a.", ".$b;
 echo $c; // si ottiene come output:
PCOPEN, una rivista di informatica
?>
```

Notate l'inserimento, tra virgolette doppie, della virgola e dello spazio vuoto per meglio visualizzare a video le due variabili.

Lo stesso risultato lo si sarebbe ottenuto utilizzando opportunamente la funzione *echo* (senza però utilizzare la variabile *$c*):

```
 echo "$a, $b"; // equivalente a: echo
$a.", ".$b;
```

Si può anche "concatenare" una stringa con se stessa con questa sintassi abbreviata:

```
<?php
 $a = "PCOPEN ";
 $a. = "amico" // equivale a: $a =
$a."amico";
 echo $a; // si ottiene a video: PC
OPEN amico
?>
```

Vi sono poi operatori aritmetici, logici e di confronto. Per comodità sono stati riassunti nel riquadro della pagina successiva, indicando solo gli operatori che verranno utilizzati nel corso; per tutti gli altri operatori (e per la loro priorità) fare riferimento al manuale di PHP.net.

Costanti

Per alcune esigenze potremmo avere bisogno di definire delle costanti e non delle variabili. In questo caso il costrutto da usare è:

```
define ("nome", valore);
echo nome;
```

Valore può essere solo di tipo scalare; nome non è preceduto dal simbolo *$* ed è case sensitive.

Applicazione: immagine random

Vediamo finalmente un esempio pratico di quanto abbiamo imparato: immaginiamo di avere inserito in una certa cartella del nostro spazio Web una serie di immagini contenute nella directory *Immagini* e aventi nome *vacanza_x.jpg* dove *x* è un numero intero progressivo che parte dal valore *1*.

Sulla nostra home page è visualizzata una di queste immagini, ma noi vorremmo, per rendere il sito più attraente e farlo sembrare sempre diverso e aggiornato, fare apparire un'immagine diversa ogni volta che l'home page viene richiesta (o quando viene fatto il refresh). La situazione di partenza è visibile nella pagina *intro.html* (*listato 8*) che mostra l'immagine (un quadrato colorato) *vacanza_1.jpg* al centro della pagina.

Dovremo fare in modo che il valore x sia generato in maniera random a ogni chiamata della home page. Utilizzeremo a tale scopo la funzione **rand(y,z)** di PHP, supponendo di avere cinque immagini nel server Web. Il risultato è visibile aprendo la pagina *random.php* (*listato 9*): la parte che ci interessa, quella che genera il numero casuale compreso tra 1 e 5 e che visualizza l'immagine è la seguente:

```
<?php
 $num=rand(1,5);
 // rand(min,max) genera un numero
casuale intero compreso tra min e max
```

inclusi
```
echo "<img src='immagini/
vacanza_".$num.".jpg'>"; //
```
gli attributi HTML vanno indicati usando gli apici singoli e non le virgolette
```
?>
```

Riprenderemo l'esempio, per approfondirlo e migliorarlo, dopo aver introdotto il costrutto **while**.

Le date

Come avrete notato, PHP non propone nessun tipo di variabile *date* o *time*, ma molto probabilmente avremo spesso bisogno di utilizzare questi formati data e tempo.

A tale scopo useremo le funzioni di data e ora previste da PHP. La principale è senza dubbio

```
date (formato, timestamp);
```

in grado di restituire tutti i possibili valori in termini di data e tempo estraendoli da uno timestamp. Il **timestamp**, nato nel modo UNIX, rappresenta il numero di secondi trascorsi dal 1/1/1970 (la cosiddetta Unix Epoch) ed è valido fino al 2037 (va bene, quindi, per le nostre esigenze attuali). Se non indichiamo il timestamp, date considererà giorno e ora di quel preciso momento del server Web ospitante PHP. Se volete inserire un vostro timestamp, dovete usare la funzione *mktime()* con questa modalità:

```
mktime (ora, minuti, secondi, mese,
giorno, anno, ora_legale) // ora_legale
(valore -1 o 0) può essere omesso:
PHP cercherà di ricavare il valore dal
sistema
```

Date fornisce una lunga lista di formati tra cui scegliere, come:

```
$timestamp = mktime (12, 0, 0, 2,
11, 2004); // timestamp del
12/02/2004, ore 12.00

echo date ("d-m-y", $timestamp);
// a video: 11-02-04
echo date ("D d, F Y", $timestamp);
// a video: Wed 11, February 2004
echo date ("H:i:s", $timestamp);
// a video: 12:00:00
echo date ("w, z", $timestamp);
// a video: 3 (giorno della settimana),
41 (giorno dell'anno)
```

Può essere interessante anche studiare la funzione **getdate**, la quale inizializza un array con tutti i valori estratti da uno timestamp specificato.

Esempi di utilizzo di date in PHP

Trattare con le date non è semplice, ma può essere molto utile saperlo fare. Ecco quindi due esempi per fare un po' di pratica.

Oggi.php (*listato 10*): serve a esprimere la data odierna utilizzando i nomi estesi dei giorni e dei mesi, in italiano. La funzione *date* non è sufficiente perché estrae sì questi dati, ma in lingua inglese. Si potrebbe usare *setlocale* e *strftime*, ma preferiamo usare un altro sistema che ci consente anche si utilizzare gli array. *Date* ci consente di estrarre il valore numerico (senza zeri) sia del giorno della settimana (0=domenica) che del mese, quindi per prima cosa creiamo due array (quello dei mesi ha il primo numero della chiave corrispondente a 1) e poi il gioco è fatto:

```
<?php
 $giorno = array ("domenica",
     "lunedì", "martedì", "mercoledì",
     "giovedì", "venerdì", "sabato");
 $mese = array (1 => "gennaio",
     "febbraio", "marzo", "aprile",
     "maggio", "giugno", "luglio",
    "agosto", "settembre", "ottobre",
     "novembre", "dicembre");
 $giorno_mese = date ("d");
 $anno = date ("y");
 $mese_nome = $mese[date("n")];
// restituisce il nome del mese
 $giorno_nome = $giorno[date("w")];
// restituisce il nome del giorno della
settimana
 echo "Ciao, oggi è $giorno_nome
    $giorno_mese $mese_nome
    $anno";
 ?>
```

counter.php (*listato 11*): serve a organizzare un elemento che funga da contatore per indicare, ad esempio, quanti giorni mancano a una data particolare. Per prima cosa dovremo ricavare il *timestamp* della data odierna (in un orario x) e della data da raggiungere (sempre allo stesso orario x. Qui prendiamo come riferimento il 31 dicembre del 2004). Facendo la differenza tra questi timestamp otterremo il numero di secondi che separa le due date, quindi dividendo per il numero di secondi di un giorno (60*60*24 = 86.400 secondi) avremo il numero di giorni che ci separano dal nostro riferimento. I dettagli sono commentati nel listato:

```
<?php
 $inizio = mktime(12,0,0,date("n"),
      date("j"), date("Y"), 0);
 $fine = mktime
      (12,0,0,12,31,2004,0);
 $diff=($fine-$inizio)/86400;
 echo "Alla fine del 2004 mancano
      $diff giorni";
?>
```

Operatori aritmetici

Poniamo: **$a=5** e **$b=3**

```
$c=$a+$b;        // somma: $c=8
$c=$a-$b;        // sottrazione: $c=2
$c=$a*$b;        // moltiplicazione: $c=15
$c=$a/$b;        // divisione: $c=1,66
$c=$a%$b;        // modulo: $c=2
```

Pur non essendo un operatore va ricordato anche:

```
$c=int($a/$b);   // quoziente intero della divisione: $c=1
```

Utilizzeremo anche gli operatori di incremento e decremento:

```
$c=++$a;         // $c=6  (pre-incremento) e $a=6
$c=–$a;          // $c=4  (pre-decremento) e $a=4
$c=$a++;         // $c=5  (post-incremento) e $a=6
$c=$a–;          // $c=5  (post-decremento) e $a=4
```

Operatori logici

Poniamo: **$a=true** e **$b=false**

```
$c=$a and $b;    // AND: $c=false
$c=$a or $b;     // OR: $c=true
$c=! $a;         // NOT: $c=false
```

Operatori di confronto

Useremo questi operatori coi costrutti di controllo.

```
$a == $b;        // true se $a uguale a $b
```

NB: l'operatore **==** effettua un controllo di confronto senza modificare il valori di $a e $b, mentre l'operatore **=** effettua un'assegnazione di valore.

```
$a <> $b;        // true se $a diverso da $b
$a < $b;         // true se $a minore di $b
$a > $b;         // true se $a maggiore di $b
$a <= $b;        // true se $a minore o uguale a $b
$a >= $b;        // true se $a maggiore o uguale a $b
```

4 I form

Per interagire con chi accede alle nostre pagine abbiamo a disposizione, tramite HTML, una serie completa di campi da compilare. L'utente ci fornirà quindi i **dati** che verranno usati come base per elaborare le azioni successive.

Come webmaster dobbiamo compiere questi passaggi: creare il form di inserimento dati, acquisire i dati (registrandoli nelle forme opportune ed eventualmente validandoli), fornire una risposta all'utente. La creazione del form si effettua con HTML e la si dà per acquisita (vedi corso webmaster). Fondamentale per passaggio, registrazione e utilizzo dei dati è la prima riga dell'istruzione form, e in particolare la parola **action** e i due metodi **get** e **post**:

```
<form action="test.php"
      method="post">
```

Questa riga indica, una volta sottoscritti i dati con l'apposito controllo *submit*, che il browser verrà indirizzato alla pagina *test.php* sul CD (se action è omessa la pagina cui l'utente viene reindirizzato è la pagina stessa che comprende il form) e i dati passeranno col metodo post.

Il **metodo post** consente di trasmettere, in maniera totalmente trasparente all'utente, tutti i dati compilati nel form senza alcun limite di spazio, mentre il **metodo get** trasferisce i dati visualizzandoli sulla riga di indirizzo del browser (e col limite di soli 256 caratteri trasmissibili). Per vedere la differenza fate riferimento a *post.php* (*listato_12,* è un form con metodo post) e a *get.php* (*listato_13*, è un form con metodo get): entrambi hanno come attributo action la pagina *test.php*, ma la differenza di trasmissione dei dati la notate sulla riga del browser (vedi *immagini 2 e 3*).

Tre istruzioni

Una volta trasferiti i dati, dovremo essere in grado di accedervi: PHP mette a disposizione queste **tre istruzioni** "superglobali":

```
$_GET["dato"] // per form con
              metodo get

$_POST["dato"] // per form con
               metodo post

$_REQUEST["dato"]  // sia per form
                   con metodo get che post
```

$_REQUEST concede una possibilità che *$_GET* e *$_POST* non danno: una volta registrato un dato del tipo *$_REQUEST["nome_dato"]* possiamo poi richiamarlo nella pagina come fosse una variabile appena dichiarata, ossia con la forma:

$nome_dato

Metodo GET

Metodo POST

Conto alla rovescia modificato

Ora siamo in grado di interagire col nostro utente. Potremmo, ad esempio, aumentare l'utilità del "conto alla rovescia" presentato nella sezione precedente, consentendo all'utente di indicare lui stesso la data cui fare riferimento.

Dobbiamo solo stare attenti ai dati inseriti (sia in termini di formato che di valori consentiti) e quindi useremo dei controlli con caselle combinate (le liste a discesa), come nell'esempio *inserisci.html* (*listato 14*). I valori inseriti saranno passati attraverso il metodo post alla pagina *conto_rovescia.php* sul CD. È possibile anche inserire date senza senso (ad esempio 31 giugno): la funzione timestamp si occuperà di trasformare l'errato inserimento nella data più vicina e visualizzeremo questa data nella risposta. La data inserita dall'utente nel form viene così riportata, tramite *$_POST*, nella pagina indicata da action:

```
// $fine è il timestamp della data
richiesta
$fine = mktime (12,0,0,$_POST[
      'mese'],$_POST['giorno'],
      $_POST['anno'],0);
```

Sarebbe comunque da impedire l'inserimento di date non valide, ma per fare questo dobbiamo aspettare di aver introdotto i costrutti di controllo.

Modifica dello stile di una pagina Web

Un altro interessante esempio lo possiamo immaginare sfruttando le caratteristiche dei **fogli di stile**: vogliamo, infatti, dare la possibilità al nostro utente di visualizzare i nostri articoli riportati su pagina Web con carattere, dimensioni e colori preferiti.

Faremo transitare il nostro utente dalla pagina *introduzione.html* (*listato_15*) dove è presente un form attraverso il quale si dovrà scegliere il carattere poi visualizzato nella pagina *articolo.php* sul CD.

La scelta dei caratteri è ridotta alle quattro famiglie (arial, verdana, times, courier) generalmente visualizzate da tutti i browser, e i colori consentiti sono espressi in termini esadecimali, anche se l'utente vede delle descrizioni "normali" (bianco, nero, giallo, ...). Scelta la combinazione voluta, la pagina *articolo.php* verrà visualizzata di conseguenza tramite le indicazioni inserite nel foglio di stile incorporato:

```
<style>
span.font {
font-family: <?php echo
      $_POST['carattere']; ?>;
font-size: <?php echo
      $_POST['dimensione']; ?>px;
color: <?php echo $_POST['col_car']; ?>;
}
body {
background-color: <?php echo
      $_POST['col_sfondo']; ?>;
}
</style>
```

POST e GET in PHP

Fino alla versione 4.2.0, PHP concedeva sempre e comunque di richiamare i dati inseriti nei form con la forma *$nome_dato*. Ora questa possibilità, salvo il caso visto con *$_REQUEST*, non è più concessa per motivi di sicurezza. Per capirne le implicazioni, immaginate che qualcuno avesse chiamato la pagina *test.php* senza passare attraverso i form: semplicemente scrivendo http://www.xxxxx.it/test.php?nome=topolino questo utente smaliziato sarebbe riuscito ad attribuire alla variabile *$nome* il valore *topolino*. Vi sono casi in cui questa possibilità conduce a eventi assolutamente indesiderati, compreso l'accesso a pagine protette.
Per utilizzare il "vecchio" sistema bisogna fare una modifica su *php.ini*, sostituendo il valore *off* a *on* sulla riga *register_globals* (il valore valido per il vostro sistema lo potete vedere anche controllando la pagina visualizzabile usando la pagina *phpinfo.php*).
Sottolineiamo però, ancora una volta, come sia meglio comunque abituarsi a usare *$_POST* e *$_GET*, sia per evitare problemi di sicurezza, sia per evitare che il vostro provider, magari aggiornando il parser PHP, cambi il valore di *register_globals* da *on* a *off* lasciandovi un bel po' di problemi sulla visualizzazione del vostro sito!

L'esempio può essere ampliato coinvolgendo tutte le caratteristiche della pagina come indicato dalle proprietà dei CSS. Anche qui, però, mancano le strutture di controllo: cosa succede se si accede alla pagina senza passare attraverso l'introduzione? Sarà il browser a decidere la visualizzazione, ma chiaramente questa è una condizione inaccettabile per il webmaster.

Accessibilità dei siti Web

Nell'esempio indicato è possibile variare la dimensione (in pixel) del carattere.

Si potrebbe pensare che questa indicazione definisca la dimensione del carattere in maniera fissa, indipendente dal browser.

In realtà se aprite la pagina *articolo.php* con Mozilla (oppure Opera o Firefox) vedrete che usando le combinazioni *CTRL+ +* o *CTRL+ -* i caratteri aumenteranno o diminuiranno di dimensione.

Microsoft Explorer, invece, non modifica assolutamente la dimensione dei font cui è stata assegnata una dimensione fissa (in punti o pixel).

Le disposizioni del W3C

Chi sbaglia? Diversamente da quanto si potrebbe pensare, è Explorer a non rispettare i dettami stabiliti per i browser Internet: anche se questa possibilità di ridimensionamento manda in crisi i webmaster (desiderosi di mantenere il layout studiato per la pagina), essa è espressamente richiesta dal W3C (*World Wide Web Consortium*) per accrescere l'usabilità del Web a persone cui può essere assolutamente necessario scalare i caratteri.

Tenete poi presente che una certa dimensione in pixel può essere splendida per certe risoluzioni video, ma per altre può rivelarsi non adatta perché troppo piccola.

Non prendetevela, quindi, se dopo aver speso giorni e giorni a stabilire la giusta visualizzazione della pagina fissando tutti i font, un semplice *CTRL+ +* potrà scompaginare tutto.

5 Costrutti di controllo

Per costruire script avanzati bisogna padroneggiare assolutamente le strutture di controllo, ossia quelle istruzioni che consentono di eseguire delle azioni solo a fronte della verifica di particolari condizioni.

L'esempio tipico è dato dalla pagina protetta da password: volendo descrivere a parole l'azione connessa, essa suona come *Accedi alla pagina solo se la password che hai fornito corrisponde a quella registrata per il nome utente inserito*. "Solo se" è il costrutto di controllo.

PHP ci fornisce tutti i costrutti standard (*for, if, while, switch*) e un paio di istruzioni estremamente utili (*foreach* e *isset*) per accelerare i tempi di scrittura del codice.

If – else – elseif

La **condizione if** (con l'eventuale aggiunta di **else** o **elseif**) è la base dei costrutti di controllo. Viene definita una condizione: se (if) è rispettata (condizione vera) viene eseguito il codice tra le parentesi graffe dopo la if, altrimenti si esce dalla if senza eseguire il codice o viene eseguito del codice alternativo (else):

```
if (condizione) {
 codice da eseguire se condizione
è vera
}
else {
 codice da eseguire se condizione
è false
}
```

Le condizioni if possono essere nidificate senza problemi per creare strutture complesse. La condizione elseif consente di aggiungere ulteriori condizioni da verificare prima di arrivare al codice else o di uscire dal costrutto.

Un esempio di utilizzo lo possiamo vedere in *confronto.php* (*listato 16*): creiamo due numeri interi casuali *$a* e *$b* compresi tra 1 e 4 e verifichiamo quale dei due sia più grande:

In alcuni frangenti si dimostra utile usare la parola chiave *exit* all'interno di una condizione: quando viene incontrata questa istruzione, il server blocca la "traduzione" della pagina e la invia al browser client come definita in quel momento. Un esempio è disponibile in *controllo-exit.php* (*listato_17*): se la prima condizione è soddisfatta non posso più visualizzare quanto scritto dopo la chiusura del ciclo if:

```
    if ($a>$b) {
    echo "\$a=$a e \$b=$b:<br>";
    echo "\$a è maggiore di \$b";
    echo "<p><h3> Verificata questa
condizione blocco il codice successivo
usando exit.
Non potete sapere cosa c'è scritto
dopo il blocco if!</h3>";
    exit;
    }
```

While

Il costrutto del ciclo **while** è veramente semplice e al contempo potente: le istruzioni contenute nel ciclo sono iterate finché la condizione è vera, ogni volta andando a leggere (ordinatamente) il valore successivo della condizione. Quando la condizione diventa falsa il ciclo si arresta e prosegue oltre:

```
while (condizione) {
 codice
}
```

Questo costrutto è molto utilizzato per leggere le liste: ad esempio si usa while per mandare a video tutti i record di una tabella terminando l'esecuzione quando la tabella è vuota (vedremo questo utilizzo nelle lezioni dedicate a MySQL).

Con il costrutto while siamo finalmente in grado di migliorare l'esempio creato per la visualizzazione random di una foto (*random.php*, *listato_09*).

Cosa succede, infatti, se inseriamo altre immagini nella cartella Web che abbiamo scelto, sempre seguendo le stesse logiche di numerazione?

Dovremmo correggere off-line il listato di cui sopra, sostituendo al valore 5 il valore più alto delle nuove immagini inserite, per poi trasferire di nuovo la pagina sul server sovrascrivendo la precedente. Facile, ma è una possibile fonte di errori: PHP ci può risparmiare questa operazione rendendo la nostra pagina valida qualunque sia il numero di immagini presenti nella cartella.

Per questo sfruttiamo le funzioni per le directory (in particolare *opendir* e *readdir*) e i costrutti di controllo.

Innanzi tutto usiamo *opendir* per registrare (e validare) il percorso della cartella, e poi *readdir* per leggere il primo file della directory.

Qui interviene while: attraverso una condizione while su *readdir*, la directory viene letta file dopo file e ogni riferimento è registrato su un array. Esau-

LISTATO 16

```
<?php
 $a = rand (1,10);
 $b = rand (1,10);
    if ($a>$b) {
       echo "$a=$a e \$b=$b:<br>\$a è maggiore di \$b";  // il carattere
       escape (\) serve a visualizzare come output simboli particolari come $
       }
    elseif ($a==$b) {   // notare l'operatore di confronto ==
       echo "$a=$a e \$b=$b:<br>\$a è uguale a \$b";
       }
    else {
       echo "$a=$a e \$b=$b:<br>\$a è minore di \$b";
       }
 ?>
```

rita la lettura, il codice esce dal ciclo while e possiamo finalmente definire $max, ossia il numero di immagini che abbiamo trovato nella directory *Immagini*.

Ecco quindi la pagina *random-due.php* (*listato 18*).

La parte più utile da analizzare è la seguente:

```
while (false !== ($file = readdir($cartella))) {
    // il ciclo while verrà iterato finché
la cartella non sarà stata interamente
letta. A ogni iterazione $file assume
il nome del file successivo.
Con la prima iterazione creo l'array
e registro il primo nome di file,
poi l'array viene riempito con i nomi
dei file seguenti
    $lista[ ]=$file;
    }
    // count conta il numero
di elementi contenuti nell'array. Viene
diminuito di due perché l'array della
cartella contiene sempre le indicazioni
UNIX "." e ".."
    $max=count($lista)-2;
    if ($max==0) {
    echo "Attenzione: Non ci sono
          immagini nella cartella";
    }
 else {
    $num=rand(1,$max);
 }
```

Ricordiamo che PHP 5, al momento rilasciato solo come release candidate, prevede una nuova funzione, **scandir**, in grado di evitare questo passaggio con while: scandir sarà infatti in grado di leggere i contenuti di una cartella e registrarli direttamente in un array senza utilizzare altre funzioni.

For

Il ciclo for consente di iterare un'azione, definendo la condizione iniziale di un puntatore, il controllo da fare e l'incremento del puntatore dopo ogni ciclo. L'azione verrà eseguita (e rieseguita) finché il controllo darà risposta true alla verifica.

```
for (punto iniziale; condizione da
rispettare; tipo di iterazione) {
    codice da eseguire
 }
```

For si rivela utilissimo per costruire strutture di pagina di cui non possiamo conoscere con anticipo tutti gli elementi: stiamo parlando, quindi, di veri e propri siti dinamici.

Per capire meglio, possiamo costruire un esempio basandoci su quanto visto per il costrutto while con la directory *Immagini*.

In questo caso, però, il nostro scopo è ottenere la lista delle immagini contenute nella cartella, visualizzandole nella pagina Web *elenco-dir.php*. Qui trovate la parte del listato che si riferisce al costrutto **for** *(listato 19)*.

Possiamo migliorare la visibilità del nostro elenco visualizzando i file alternativamente su righe aventi colore di sfondo diverso (*elenco-dir-due.php, listato_20*). Qui l'attenzione va posta nel conteggio delle righe da mandare in output: se sono pari verrà eseguito un ciclo for e usciremo dalla pagina, mentre se sono dispari il ciclo for dovrà fermarsi prima dell'ultima riga, che andrà stampata a parte (*listato 20*).

Un buon esercizio per verificare di aver compreso bene i concetti di cui sopra può essere il seguente (è una variazione sul tema di quanto appena visto): creare una galleria delle immagini della directory, visualizzandole a due a due, una di fianco all'altra.

Chiaramente dovremo usare le tabelle e/o i CSS, ponendo attenzione all'eventuale ultima immagine dispari. La soluzione da me proposta la trovate nella pagina *elenco-galleria.php* (sul CD).

Vedremo nelle prossime puntate del corso come creare una galleria con un certo numero prefissato di immagini, o altri oggetti, per ogni pagina (ad esempio sei immagini per pagina) e una barra di navigazione per spostarsi tra le pagine.

LISTATO 19

```
echo "<h4>La cartella 'immagini' contiene questi file:</h4><p>";
        for ($i=2;$i<=count($lista)-1;$i++) {  // il contatore $i parte da 2
            perché i primi due valori registrati nell'array $lista sono
            "." e "..", e non voglio visualizzarli

        $ordine=$i-1;
        echo "————————————————<br>";
        echo "<span class='output'>File $ordine: $lista[$i]</span><br>";
    }
```

LISTATO 20

```
if ($fine%2==0) { // controllo se nella cartella c'è un numero pari di immagini
        for ($i=2;$i<count($lista);$i=$i+2) {  // il ciclo di for visualizza due file
            ad ogni iterazione
            $ordine=$i-1;
            $j=$i+1;
            echo "<span class='dispari'>File $ordine: $lista[$i]</span>";
            echo "<span class='pari'>File $i: $lista[$j]</span>";
            }
        }
 else { // il numero di immagini è dispari, quindi si deve aggiungere l'ultima riga
         dispari
        for ($i=2;$i<$fine;$i=$i+2) { // il ciclo for visualizza due file per ogni
        iterazione e si ferma prima  dell'ultima immagine
            $ordine=$i-1;
            $j=$i+1;
            echo "<span class='dispari'>File $ordine: $lista[$i]</span>";
            echo "<span class='pari'>File $i: $lista[$j]</span>";
            }
        $j=$i-1;
        echo "<span class='dispari'>File $j: $lista[$i]</span>";
        }
```

Switch

Il costrutto **switch** somiglia a una serie di if di uguaglianza: una variabile (array e object esclusi) viene confrontata con dei valori predefiniti così da eseguire differenti blocchi di codice a seconda del valore assunto dalla variabile:

```
$mese=date('n');
switch ($mese) {
 case 1:
    echo “Siamo in Gennaio”;
    break;
 case 2:
    echo “Siamo in Febbraio”;
    break;
 case 3:
    echo “Siamo in Marzo”;
    break;
 .....
 }
```

Lo stesso risultato si sarebbe raggiunto con:

```
 $mese=date('n');
 if ($mese==1) {
    echo “Siamo in Gennaio”;
 }
 if ($mese==2) {
    echo “Siamo in Febbraio”;
 }
 if ($mese==3) {
        echo “Siamo in Marzo”;
 }
 .....
```

Dove sta la differenza? Nel primo caso la variabile $mese è valutata all'inizio del costrutto e poi il valore è solo confrontato con quelli delle istruzioni case, mentre nel caso di **if** la variabile $mese è valutata e verificata ogni volta, con uno spreco di risorse e tempo di elaborazione.

Notate la parola chiave *break*: va inserita al termine di ogni blocco di codice case, altrimenti il parser continuerebbe leggendo il blocco successivo invece di uscire dal costrutto.

È possibile definire un case con valore default: il codice sarà eseguito quando tutte le altre condizioni case si riveleranno false.

Isset

Abbiamo spesso la necessità di controllare, prima di compiere un'azione, se una variabile è definita (anche se magari con valore nullo) oppure se non lo è.

Questo controllo è assolto semplicemente da isset in combinazione con if: isset, infatti, restituisce un valore true se la variabile cui fa riferimento esiste:

```
 if (isset($variabile)) {
    codice
 }
```

Possiamo usare isset per migliorare l'esempio in cui modificavamo la visualizzazione di un articolo. ▷

LISTATO 21

```
<style>
 span.output {
    font-family:
    <?php
       if (isset($_POST['carattere'])) {
          echo $_POST['carattere'];
          // valido solo se è stato sottoscritto il form e quindi $_POST esiste
       }
       else {
          echo "Arial";
          // è la visualizzazione di default quando $_POST non esiste,
          ossia non è mai stato cliccato il bottone “invia” del form
       }
    ?>;
 }
 </style>
```

LISTATO 22

```
$lista=array (“pippo”, “topolino”, “paperino”, “pluto”);
 foreach ($lista as $copia) {
    echo “Un personaggio Disney: $copia<br>”;
 }
```

Si può utilizzare foreach anche per definire un secondo array contenente le chiavi dell'array originale:

```
 $lista=array (“pippo”, “topolino”, “paperino”, “pluto”);
 foreach ($lista as $chiave=>$copia) {
    echo “Personaggio Disney numero $chiave -> $copia<br>”;
 }
```

LISTATO 23

```
if (!isset($_POST['nome'])) {
 // serve solo alla prima chiamata della pagina
       die ("<p><h3>Tutti i campi vanno obbligatoriamente compilati</h3>");
    }
 foreach ($_POST as $ctr) {
 // foreach ci consente di controllare tutto l'array $_POST
    if (trim($ctr)==""){
    // trim toglie gli spazi vuoti eventualmente presenti in $ctr
       die ("<p><h3>Tutti i campi vanno obbligatoriamente compilati</h3>");
    }
 }
```

Invece di usare un form intermedio di passaggio (tra l'altro poco elegante, dal momento che il nostro utente potrebbe accedere direttamente alla pagina *articolo.php* una volta che ne conosca l'URL), potremmo definire dei valori di default validi finché non viene scelto un diverso carattere dal form cliccando su “invia”.

Il controllo della definizione iniziale delle variabili è demandato proprio a isset come esemplificato in *articolo-due.php* (*listato 21*):

Foreach

Foreach, un po' come isset, ci aiuta a risparmiare tempo per compiere un'azione di cui spesso si ha bisogno: attraversare completamente un array, effettuando delle operazioni (assegnazione, confronto, visualizzazione, ...) su un secondo array “copia” appositamente creato.

La funzione più importante la si rileva quando bisogna operare su un array completo, anche solo per visualizzare tutti gli elementi.

Per fare questo potremmo usare un ciclo for (ma dobbiamo sapere quanti elementi compongono l'array) o un ciclo while (con una struttura non molto amichevole). Oppure un semplice ciclo foreach (pagina *lista-disney.php*, *listato 22*):

Un altro esempio concreto lo si ha in occasione della validazione dei dati passati attraverso un form: se vogliamo porre tutti i campi di un form come obbligatori, dovremo controllare tutti i campi per verificare che sia stato immesso un valore, inserendo quindi una serie di istruzioni if e/o isset. Foreach ci viene in aiuto per rendere la scrittura più agile e per evitare banali dimenticanze, come si vede dalla pagina *check.php* (*listato 23*):

Notate, nel ciclo if, la funzione **trim** il cui scopo è eliminare tutti gli spazi vuoti all'inizio di un campo: in questo modo se un utente compilasse il form solo con degli spazi vuoti, il nostro test comunque non ne valiederebbe l'inserimento (una funzione if semplice, invece, l'avrebbe fatto passare).

Un controllo di questo tipo è sufficiente per la maggior parte delle nostre esigenze, come vedremo nella prossima puntata del corso.

Introduciamo qui anche la funzione **die** che interrompe l'esecuzione della pagina visualizzando come output il codice chiuso dalle parentesi.

Lo stesso compito era assolto anche da exit, vista in uno degli esempi precedenti.

Siti di riferimento

Se volete avere ulteriori approfondimenti su PHP potete visitare questi siti:

http://www.php.net
http://freephp.html.it
http://www.apache.org

Se state ricercando un suggerimento su una qualsiasi funzione di PHP, potete accedere velocemente alle pagine del manuale on-line scrivendo l'URL del sito seguito da uno slash con la parola da ricercare, ad esempio http://www.php.net/isset (sul manuale sarà cercata la parola isset).

È utile farlo perché, oltre alla descrizione del manuale, si trovano anche molti esempi pratici scritti dagli utilizzatori di PHP. ■

I corsi Webmaster disponibili nel CD

Nel *CD Guida 2* allegato a questo numero di *PC Open*, all'interno della cartella *PDF/Corsi*, trovate due corsi completi che possono essere un utile complemento al corso PHP. Uno è il corso *Web Developer ASP*, 97 pagine suddivise in quattro lezioni per capire come realizzare siti dinamici in tecnologia ASP.

Il corso *Webmaster* spiega, invece, in 88 pagine suddivise in otto lezioni tutto quello che bisogna sapere per costruire un sito e imparare il linguaggio HTML 4.01, i CSS (fogli di stile), Java Script e CGI. Il corso è completato da utili consigli per promuovere il proprio sito on line.

▶ A scuola con PC Open

Web Developer PHP

di Federico Pozzato

1 Approfondire PHP

Nella lezione 1 abbiamo esplorato le basi di PHP, dando tutti gli elementi per creare le nostre prime pagine e alcuni spunti per cogliere le potenzialità di questo linguaggio.

Nella seconda lezione metteremo in luce, invece, funzionalità avanzate che ci consentiranno di migliorare i nostri script, rendendoli più efficienti e utilizzabili anche per il futuro. Il discorso sarà poi concluso nella terza puntata, dove verranno trattate anche tematiche relative alla sicurezza e alla gestione degli errori.

Non parleremo ancora della connessione tra PHP e i database (con particolare riguardo a MySQL), in quanto questo argomento sarà oggetto di trattazione specifica e approfondita nelle prossime puntate del corso.

IL CALENDARIO DELLE LEZIONI

Lezione 1:
- PHP con un server off line
- Funzioni base e variabili
- I costrutti di controllo

▶ Lezione 2:
- **Approfondiamo PHP**
- **Include e require**
- **Funzioni e classi**
- **Proteggere una pagina**
- **Cookie e sessioni**
- **La funzione mail**

Le prossime puntate

Lezione 3: PHP e database
Lezione 4: PHP e MySQL
Lezione 5: Gestire un sito dinamico con PHP e MySQL

2 Include e require

I listati completi sono sul CD

Spesso capita di inserire in più pagine Web esattamente lo stesso codice: un esempio tipico sono le righe che formano la testata (*header*) o il piè di pagina (*footer*), ma possiamo prendere in considerazione anche le righe che compongono le barre di navigazione e dei menu, le chiamate ai database, la protezione di pagine con password, la chiamata dei CSS esterni e altro ancora.

Come agiamo in questi casi? Usando il "tradizionale" HTML, non abbiamo molta scelta: ci dobbiamo armare di pazienza ed effettuare una serie di copia-incolla di codice su pagine diverse! A parte gli errori sempre in agguato nell'operazione del copia-incolla, un grosso problema nascerà con la necessità di modificare una parte qualsiasi di questo codice ripetuto: prima o dopo, inutile negarlo, avremo bisogno di cambiare qualcosa, vuoi per migliorare le funzionalità e la navigazione, vuoi per aggiornamenti e refresh. Di fronte a questa esigenza dovremo nuovamente armarci di pazienza (molta più di quella che ci era servita quando abbiamo realizzato le pagine), aprire tutti i file interessati ed effettuare la modifica. L'operazione può essere più o meno agevole a seconda del tipo di cambiamento e del software utilizzato, e può divenire un ostacolo insormontabile se i listati delle pagine sono stati creati da terzi. Alcuni programmi, come Home Site, ci vengono in aiuto con una utilissima funzione che consente di operare il replace su più file contemporaneamente, però questo non è sufficiente.

Leggendo queste righe immaginiamo che qualcuno abbia pensato ai fogli di stile (CSS, *Cascading Style Sheet*) e ai problemi di modifica del layout di un sito Web. Prima dell'introduzione dei CSS, infatti, modificare gli attributi di struttura (anche solo la dimensione di una font) su tutte le pagine di un sito Web era un lavoro improbo, mentre adesso basta semplicemente cambiare un unico attributo in un CSS esterno e il gioco è fatto.

Include e **require** in PHP funzionano allo stesso modo di un CSS esterno: nel punto voluto del listato Web possiamo inserire una chiamata a un file esterno il cui contenuto sarà "trasferito" alla pagina chiamante e interpretato dal parser PHP come appartenente alla pagina di origine. Ogni eventuale modifica andrà effettuata, quindi, solo sul file esterno e automaticamente verrà estesa a tutte le pagine interessate.

PHP ci propone due funzioni simili: *include* e *require*. Si inseriscono in maniera estremamente semplice nella pagina chiamante, semplicemente indicando il nome (col percorso relativo) del file voluto:

```
<?php
   include ("testata.inc.php");
?>

<?php
   require ("testata.inc.php");
?>
```

La differenza tra le due funzioni è nella gestione degli errori: in caso di errore, *include* produce solo un avvertimento (*warning*) senza bloccare la pagina, mentre *require* blocca la compilazione del linguaggio (*fatal error*). Che cosa usare, quindi, è funzione degli obiettivi del Webmaster.

Un esempio semplice del vantaggio di usare gli include/require file lo possiamo ricavare direttamente dalla quotidianità. Immaginiamo di aver creato un sito dove sia necessario inserire in molte pagine un disclaimer a tutela del trattamento dei dati personali, come previsto dalle norme sulla privacy. Essendo Webmaster previdenti abbiamo incluso in ogni pagina interessata (come *privacy.php*, *vedi listato_01*) un file esterno col testo del disclaimer (*disclaimer.inc.php*, *vedi listato 2*):

```
<html>
   <head>
      <title>Privacy</title>
   </head>
```

```
  <body>
  <h3>Questa parte di testo è inserita
direttamente nella pagina
privacy.php</h3>
bla bla bla bla bla bla bla
<p>
<h3>Quella che segue, invece, è
inserita nella pagina disclaimer.inc.php
ed è visualizzata in questa pagina
utilizzando un'istruzione include:</h3>
<?php
   include("disclaimer.inc.php");
?>
   </body>
</html>
```

listato 1

```
<em>
Ai sensi della L. 675/96 ti informiamo
che tutti i dati in nostro possesso
verranno utilizzati solo per
pubblicizzare iniziative inerenti XYZWK.
Nessun dato verrà ceduto a terzi. In
ogni momento potrai chiedere di
essere cancellato dal nostro database
con una semplice comunicazione al
nostro indirizzo di posta elettronica.
</em>
```

listato 2

Quando il browser chiamerà *privacy.php*, l'effetto sarà di vedere riportato il testo del disclaimer come se appartenesse alla pagina originale *(immagine 1)*. Il vantaggio di questa struttura lo avremmo tangibilmente verificato all'inizio del 2004 quando è entrata in vigore la nuova norma di disciplina del settore (dalla Legge 675/96 al D.Lgs.196/03): invece di modificare tutte le pagine, l'unica operazione di modifica l'avremmo compiuta solo su *disclaimer.inc.php* semplicemente modificando il riferimento della Norma (vedi l'esempio corrispondente *privacy_new.php* con *disclaimer_new.inc.php*).

Bisogna porre molta attenzione al fatto che le pagine cui fanno riferimento *include* e *require* sono a tutti gli effetti pagine appartenenti al sito e richiamabili col browser. Se inseriamo note particolari in questi file esterni (magari anche password), dobbiamo essere sicuri che non siano visibili nel caso qualcuno ci finisca dentro direttamente. Il consiglio, quindi, è di nominare questi file inclusi come *nomepagina.inc.php*, dove il suffisso *inc* serve a noi per capire al volo che quella pagina è inclusa in un'altra, mentre l'estensione *php* indica al server di trattare la pagina utilizzando il parsing (è ovvio che nel listato dovranno essere presenti i tag PHP).

Se usassimo solo l'estensione *.inc* (o *.html* o *.txt* o nessuna) il browser ci farebbe vedere la pagina come fosse un testo, anche se contenesse i tag PHP. Buona norma è dunque provare ad aprire direttamente col browser tutti i file inclusi e vedere se è tutto OK per la sicurezza dei nostri dati.

Se volete una prova del rischio che si può correre, provate ad aprire la pagina *accesso.inc*: vedrete in maniera del tutto trasparente i dati di accesso a un database, password compresa *(immagine 2)*. Provate poi a chiamare *accesso.inc.php* e vedrete la differenza!

Un esempio pratico

Se il primo esempio serviva solo a entrare nell'argomento, la seconda proposta, invece, può trovare utili applicazioni pratiche nelle pagine del nostro sito: useremo include, gli array e i costrutti di controllo per costruire una barra di navigazione dinamica.

Lo scopo è sapere sempre in quale sezione del sito ci troviamo e dare la possibilità di spostarsi velocemente nelle altre sezioni. La barra verrà inserita come *include* in tutte le pagine: sarà quindi molto flessibile per permettere modifiche future nel caso di ampliamento del progetto.

Per il nostro esempio abbiamo bisogno di una pagina *main.php*, due sottopagine *arg1.php* e *arg2.php* che contengono le sezioni "viaggi e foto" e "curriculum", una pagina *menu.inc.php* (*listato_03*) per la barra di navigazione e un foglio di stile *menu.css* per definire l'aspetto della barra stessa.

La logica da applicare è la seguente:

- inizializziamo un array assegnando a ogni nome di pagina un testo da visualizzare nella barra che verrà creata;
- verifichiamo in quale pagina ci troviamo. Per fare questo utilizziamo una delle tante variabili superglobals di PHP (in questo caso *$_SERVER["PHP_SELF"]*) in unione con la funzione basename che restituisce solo il nome del file eliminando il resto del percorso
- creiamo una struttura di controllo di tipo *foreach-if*: la funzione *foreach* ci consente di navigare lungo tutto l'array creato, mentre *if* si occupa di "distinguere" la pagina attiva (quella dove ci troviamo) dai link cliccabili.

```
<?php
// inizializzo l'array
$naviga = array ("main.php"=>"Home
page", "arg1.php" => "Viaggi e foto",
"arg2.php" => "Curriculum");

// verifico in quale pagina mi trovo
$nome_file =
basename($_SERVER["PHP_SELF"]);
echo "<table><tr align='center'><td
class='lato'> </td>";

// struttura di controllo
foreach ($naviga as $chiave=>$menu)
{
echo "<td ";
if ($chiave==$nome_file) {
echo " class='nolink'>".$menu;
}
else {
echo " class='link'><a class='nav'
href='".$chiave."'>$menu</a>";
}
echo "</td>";
}
echo "</td><td
class='lato'> </td></tr></table
>";
?>
```

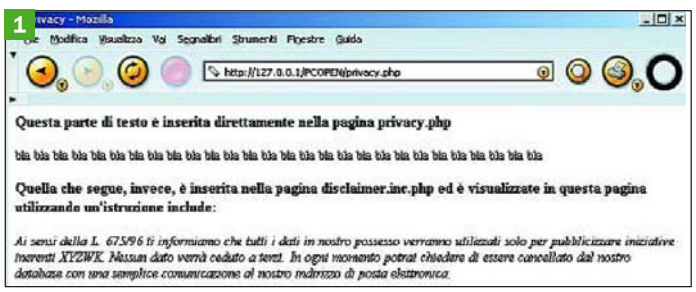
listato 3

È chiaro che, se aggiungo altre pagine, l'unica operazione da compiere sarà aggiungere la nuova pagina all'array *$naviga* nel file incluso *menu.inc.php*.

Il risultato ottenuto (uno dei tanti possibili a seconda di come definiamo il foglio di stile) è visibile nell'*immagine 3*: la pagina dove ci troviamo è visibile con sfondo verde e non è cliccabile, mentre lo sono le altre pagine. Fate clic per vedere cosa succede entrando nelle altre pagine.

La funzione *basename* consente anche di estrarre la radice del nome della pagina, eliminando l'estensione. Per fare questo, l'estensione da eliminare va indicata come secondo argomento di basename:

```
<?php
$nome_file =
basename($_SERVER["PHP_SELF"],
".php");
?>
```



Include e require *facilitano l'inserimento dei file di testo uguali in numerose pagine*

Nella pagina accesso.inc *sono trasparenti anche i dati riservati*

La pagina è visibile con fondo verde *e non è cliccabile, ma lo sono le altre pagine*

3 Funzioni e classi

Spesso si scrivono righe di codice uguali per effettuare operazioni ripetitive e comuni a più pagine. Esempi possono essere gli script per la creazione di qualche menu, ma anche listati molto più semplici con i quali si vuole magari calcolare l'area di un triangolo o restituire un valore vero-falso dopo una verifica.

Una soluzione semplice e banale può essere usare la classica accoppiata copia-incolla, soluzione valida, però, solo se si sa esattamente dove recuperare il codice da copiare. Potremmo riuscire a risolvere velocemente il nostro problema, aprendo comunque la strada a possibili errori: copiatura errata del codice, difficoltà ad adattare il nome delle variabili con possibili dimenticanze sempre in agguato, propagazione di eventuali errori contenuti nelle righe originali, difficoltà nell'individuazione del codice che potrebbe essere diversamente incapsulato nelle pagine.

Per venire incontro alle esigenze dei Webmaster, PHP ci aiuta con funzioni e classi.

Le funzioni

Le funzioni sono script di codice da invocare e da cui ci si attende una risposta:

```
function esempio (eventuali
argomenti)
{
   codice
   return risultato restituito
}
```

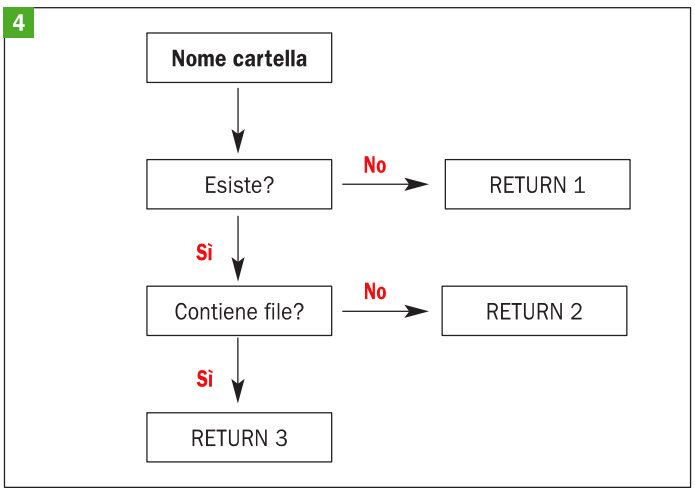
La funzione deve avere un nome univoco (attenzione: non possiamo utilizzare i nomi riservati previsti da PHP, come ad esempio *print*) e va inserita (sembra una cosa ovvia, ma non sempre è rispettata) in un punto del listato precedente alla chiamata. Il risultato viene restituito utilizzando l'istruzione *return*: chiamata all'interno di una funzione, *return* termina immediatamente l'esecuzione della funzione corrente e restituisce il suo argomento come valore della funzione. Possiamo inserire più istruzioni *return* all'interno della stessa funzione, tenendo presente che, appena incontrato il *return*, il codice proseguirà al di fuori della funzione.

Finora abbiamo visto moltissime funzioni predefinite di PHP: *basename*, ad esempio, è una di queste. Basename, infatti, necessita di alcuni argomenti (il percorso della pagina e un'eventuale estensione) e restituisce di conseguenza il valore atteso, ossia il nome della pagina con o senza estensione.

Una funzione può restituire anche solo un valore vero-falso (*true-false*), da usare magari in combinazione con un *if* per creare un ciclo di controllo, o un qualsiasi altro tipo di variabile (valori, array, oggetti).

Usare le funzioni consente di concentrarsi sul codice puro ("astratto") della funzione, senza doversi preoccupare del significato delle variabili all'esterno della funzione stessa.

4



Un algoritmo esprime la logica *che sta alla base della funzione* controllo ()

Una volta scritta, la funzione sarà sempre utilizzabile e non dovremo neppure preoccuparci di reinterpretare il codice per adattarlo alle nuove variabili. Spesso passa molto tempo tra quando si scrive del codice e quando poi lo si riutilizza: anche in questo caso, con le funzioni evitiamo di doverci rinfrescare la memoria ricompiendo i passaggi logici del passato.

Basename è registrata nelle librerie di PHP ed è valida universalmente per tutti gli utilizzatori. Nel nostro piccolo, però, anche noi abbiamo bisogno di semplificarci la vita e quindi decidiamo di scrivere la nostra prima funzione *(listato 4)* per calcolare il prezzo netto di un bene, dato uno sconto espresso come valore da 0 a 100:

```
function prezzo_netto($prezzo,
$sconto)
{
if ($sconto<0 or $sconto>100) {
return "Questa funzione assume che lo
sconto debba essere un numero
compreso tra 0 e 100, mentre tu hai
inserito uno sconto uguale a $sconto";
}
$netto=$prezzo*(1-$sconto/100);
return $netto;
}
```

listato 4

Questa funzione può essere usata dovunque ce ne sia bisogno: l'applicazione pratica la si vede nella pagina *sconto.php* che contiene un form con due campi dove inserire i valori di prezzo e sconto sulla base dei quali calcolare il prezzo netto. Da notare è la chiamata della funzione *prezzo_netto* nella pagina *sconto.php*:

```
echo "Prezzo netto: ".prezzo_netto
($_POST['pre'],$_POST['sco']);
```

Come si vede, le variabili passate come argomento della funzione hanno un nome diverso da quello assegnato nello script della funzione, ma la cosa non ha la minima importanza perché questo è il senso delle funzioni: PHP, infatti, interpreta la prima variabile passata come fosse la variabile *$prezzo originale* e la seconda variabile come fosse la *$sconto originale*.

Va sottolineato che le variabili usate all'interno della funzione non possono essere richiamate all'esterno, quindi qualsiasi invocazione a *$prezzo* o *$sconto* nel corso del listato non produrrà alcun risultato (salvo, ovviamente, non esistano nel corpo della pagina due variabili con questo nome).

Se volessimo questa possibilità dovremmo definire le variabili della funzione come *$GLOBALS* (col loro nome) all'interno dello script della funzione stessa. L'esempio è riportato nella pagina *sconto_global.php* dove, all'interno della dichiarazione della funzione *prezzo_netto* è presente l'istruzione:

```
$GLOBALS["sconto"] = $sconto;
```

Adesso sarà possibile chiamare la variabile *$sconto* anche all'esterno della funzione, mentre non si potrà chiamare *$prezzo* perché non è stata dichiarata globale.

Possiamo definire anche fUnzioni senza argomenti: è il caso delle funzioni che effettuano dei controlli ripetitivi e restituiscono una stringa o dei valori true/false.

La funzione *controllo()*, proposta come esempio, verifica per prima cosa se esiste una cartella *varie* posta allo stesso livello della pagina Web su cui viene inserita. Se la cartella esiste, viene poi verificato se è vuota o se contiene dei file. La logica che sta alla base della funzione è esprimibile in forma di algoritmo, come si può vedere nell'*illustrazione 4*. Per funzioni semplici, disegnare un algoritmo può essere solo una perdita di tempo, ma è essenziale per funzioni più complesse ed è quindi buona abitudine imparare a usarli.

La funzione è riportata nel *listato 5*, mentre la pagina Web di esempio è la *checkvarie.php*. Provate a creare e cancellare la cartella *varie*, lasciandola vuota o inserendo qualche file.

```
function controllo()
// questa funzione controlla che in
una directory predefinita sia presente
```

```
almeno un file
{
$cartella=@opendir('varie');
if (!$cartella) {
return "Attenzione: è inutile cercare
file... la cartella non esiste e quindi
devi prima crearla";
}
while (false !== ($file =
readdir($cartella))) {
$lista[]=$file;
}
$fine=count($lista)-2;
if ($fine==0){
return "Attenzione: Non ci sono file
nella cartella";
}
return “Prosegui pure: nella cartella
c'è almeno un file”;
}
```

listato 5

Per rendere più utilizzabile questa funzione, si può passare alla funzione *un argomento uguale al percorso della cartella da controllare*. La modifica è facile (la trovate in *checkvarie_plus.php*): basta definire la **funzione controllo** (*$directory*) e inserire *$directory* all'interno della funzione come argomento di *opendir*.

Ulteriore esercizio è prendere il percorso della cartella da controllare direttamente da un form: provate a costruire la pagina da soli e poi confrontatela con *checkvarie_form.php (vedi immagine 5)*. Adesso potete controllare tutte le cartelle che volete, l'unica accortezza è ricordarsi che il percorso della cartella è relativo e non assoluto.

Abbiamo detto all'inizio del paragrafo che la funzione originale deve essere presente nel codice prima di un'eventuale chiamata. Potremmo pensare di inserire (copia-incolla) la funzione in tutte le pagine che ci interessano, ed è una valida idea se le pagine che usano questa funzione sono poche, altrimenti ricadiamo nei problemi già descritti in precedenza. Una soluzione più efficiente (ed elegante) consiste, invece, nello salvare le funzioni in file separati includendoli poi come già visto. Ogni modifica fatta sulla funzione verrà quindi propagata automaticamente a tutte le pagine:

```
<?php
include (“funzione_controllo.inc.php”);
?>
```

Non è possibile, per chiarezza di comportamento del codice, ridefinire funzioni già dichiarate all'interno di uno stesso listato (overloading).

Le classi

Le funzioni sono costrutti semplici, adatti a lavori ripetitivi da cui ci si aspetta un unico risultato.

Non sempre ciò è sufficiente, basti pensare al caso di costruzioni di codice avanzato in grado di interagire con altre funzioni e variabili: un esempio può essere la creazione di un sistema di gestione avanzato di commercio elettronico, magari da esportare su più siti. In questo caso un'architettura demandata a funzioni stand-alone è possibile, ma diventa sempre più difficile da gestire all'aumentare della complessità richiesta.

Per esigenze di questo tipo ci viene in aiuto il concetto di **classe**, grazie al quale potremo addentrarci brevemente nella programmazione rivolta agli oggetti. La trattazione non può essere esaustiva perché l'argomento prevede conoscenze specialistiche, ma la cosa importante è riuscire a cogliere il concetto alla base della programmazione OO (**Object Oriented**) per poterlo poi approfondire con testi o corsi specifici.

Orientarsi agli oggetti significa porre l'accento sulla definizione di tutte le proprietà generiche di un certo oggetto e delle funzioni legate a queste proprietà. Lo faccio senza pensare all'oggetto specifico X o all'oggetto specifico Y, bensì guardando al concetto più astratto possibile grazie al quale posso poi definire gli oggetti specifici.

Una classe è, quindi, una collezione di variabili (proprietà) e funzioni (metodi) che utilizzano queste variabili:

```
class Nuova_classe
{
variabili (proprietà)
funzioni (metodi)
}
```

Definita una classe come vedremo in seguito, possiamo generare un oggetto utilizzando il costrutto new:

```
$oggetto = new Nuova_classe;
```

È il codice della classe che “definisce” l'oggetto, il quale deve essere creato, tramite l'istruzione *new*, con un nome diverso da quello di altri oggetti istanziati (cioè inizializzati) dalla stessa classe. Per il concetto stesso che sta alla base della programmazione OO, infatti, è possibile definire quanti oggetti si vuole basandoli su una stessa classe. Ricordate la scorsa puntata quando parlavamo dei tipi di variabili? Mancava solo il tipo *object*, ossia proprio quello definito usando la classe.

Definito un oggetto, a esso è possibile applicare tutti i metodi implementati nella classe di appartenenza, in maniera molto semplice e con una pulizia di codice inarrivabile per una programmazione standard.

Facciamo un esempio tratto dalla vita di tutti i giorni: se devo disegnare un rombo grande rosso e un rombo piccolo giallo, sostanzialmente devo fare due operazioni simili.

Qual è la differenza tra i due “oggetti” da disegnare? Solo la dimensione e il colore, dal momento che la definizione di rombo è unica! Bene, allora è sufficiente che sia dichiarata (una volta per tutte) una “classe” Rombo all'interno della quale siano introdotte le variabili dimensioni e colore che consentono di effettuare il disegno.

Alla classe non interessa il tipo di rombo da disegnare, ma solo definire che cos'è un rombo colorato. A questo punto per risolvere il nostro compito non ci resta che creare due “oggetti” facenti riferimento a questa classe, inizializzando semplicemente i due oggetti con le loro caratteristiche. Siamo in grado di creare infiniti rombi e, fondamentale, siamo in grado di farlo pur non sapendo nulla di cosa sia un rombo! Forniti i dati di colore e dimensioni, infatti, è la classe che si incarica di restituire gli oggetti in maniera del tutto trasparente per l'utente.

Vediamo di fare un esempio di programmazione di classe: supponiamo di creare una classe Quadrato grazie alla quale, una volta definito il lato, possano essere ricavati i valori di area, perimetro e diagonale. La definizione completa della classe la trovate in *classe_quadrato.inc.php*, mentre qui ci basta un estratto (*listato_06*) con solo il metodo “area”.

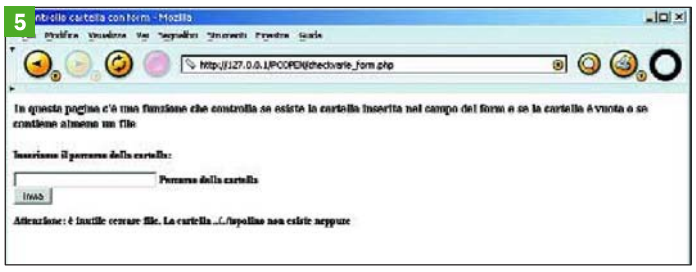
```
class quadrato
{
// dichiarazione delle variabili
utilizzate (da accompagnare con una
descrizione, per rendere più leggibile il
codice)
var $lat;
var $ar;
var $per;
var $diago;

function quadrato($lato)
// costruttore
{
   $this->lat=$lato;
}

function area()
// calcola l'area del quadrato
{
   $this->ar=$this->lat*$this->lat;
   return $this->ar;
}
}
```

listato 6

Ci sono due cose da notare: quando è definita una funzione con lo stesso nome della classe, questa funzione prende il nome di **costruttore**. Il costruttore è utile in certe classi perché inizializza il tipo di oggetto della classe stessa. In questo caso il costruttore serve a inizializzare l'oggetto col valore del lato del quadrato. Il costruttore può anche non essere indicato, ma se lo è sarà interno alla classe e avrà necessariamente il nome della classe stessa.



Il risultato in pagina *della funzione controllo*

La seconda notazione importante è il modo di passare il nome delle variabili all'interno della classe: abbiamo detto che ogni classe può creare N oggetti basati su di essa, ma sempre con nome diverso. Non potendo a priori conoscere il nome dell'oggetto, all'interno della classe dovremo usare il costrutto *$this->* generico:

```
$this->nome_variabile
```

In questo modo tutte le variabili definite nella classe saranno disponibili per ogni oggetto creato in seguito.

Un esempio completo è riportato nella pagina *classe.php*, dove possiamo vedere come inizializzare gli oggetti e come chiamare variabili e funzioni. Un oggetto va istanziato (inizializzato) usando il costrutto *new*:

```
$a=5;
$nuovo_quadrato_1=new
quadrato($a);
```

Creato l'oggetto *$nuovo_quadrato_1*, per chiamare le funzioni o le variabili useremo la notazione che già conosciamo (->), ricordandoci di indicare sempre il nome dell'oggetto appena creato *(vedi listato 7)*:

```
// creo un nuovo oggetto della classe
quadrato con lato 5
$a=5;
echo "<h3>Quadrato_1:<p></h3>";
$nuovo_quadrato_1=new
quadrato($a);

echo "<h5>Lato:
".$nuovo_quadrato_1->lat."<p>";
// per chiamare la funzione “area” e
per ritornare il valore dell'area:
echo "Area: ".$nuovo_quadrato_1-
>area()."<p>";
// per chiamare la funzione
“perimetro” e per ritornare il valore del
perimetro:
echo "Perimetro:
".$nuovo_quadrato_1-
>perimetro()."<p>";
// per chiamare la funzione
“diagonale” e per ritornare il valore
della diagonale
echo "Diagonale:
".$nuovo_quadrato_1-
>diagonale()."<p></h5>";
```

listato 7

Questo esempio di classe è banale, ma permette di cogliere l'essenza della programmazione OO: definite proprietà e metodi di una classe sarà possibile, infatti, creare e gestire oggetti in maniera semplice e senza preoccuparsi di tutto quello che c'è nella classe.

Con una ricerca su Internet si trovano numerose classi pronte da usare. Una, ad esempio, serve a superare il limite del *timestamp*: non occorre capire cosa c'è sotto, ma solo sapere come istanziare l'oggetto e quali funzioni chiamare.

In *classe.php* abbiamo inserito due oggetti diversi (due quadrati di lato diverso) basati sulla stessa classe (*$nuovo_ quadrato_1* e *$nuovo_quadrato_2*): le chiamate a variabili e funzioni differiscono solo ed esclusivamente per il nome dell'oggetto, e ci consentono di ottenere facilmente tutti i dati utili di un “oggetto” quadrato.

Lasciamo per ultima una considerazione ormai ovvia: la classe va naturalmente “inclusa” nella pagina chiamante e posta prima della creazione di qualsiasi oggetto.

Una classe può essere anche la base di un'altra classe che ne estende le funzionalità. In questo caso nella seconda classe indicheremo solo il codice nuovo senza doverci preoccupare di funzioni e variabili già costruite nella prima classe. Il costrutto da cercare nel manuale di *php.net* è *extends*:

```
class quadrato_plus extends quadrato
{
  codice
}
```

4 Proteggere una pagina con password

Con la definizione di funzioni e classi abbiamo esaurito le nozioni fondamentali di PHP.

Arrivati a questo punto non dovrebbe quindi essere un problema affrontare un compito molto pratico (e utile), ossia elaborare qualche strategia per proteggere una o più pagine del nostro sito Web da occhi indiscreti. In pratica, vogliamo che a certe pagine possano avere accesso solo le persone dotate di una password, o meglio di una precisa accoppiata username-password.

I motivi di questa decisione possono essere i più vari: su una pagina possiamo avere inserito testi e immagini che solo i nostri amici più cari devono leggere, ma potrebbe anche essere una esigenza di lavoro che ci impone di tutelare certe informazioni, oppure una parte del Web site deve essere protetta perché da essa è possibile gestire gli aggiornamenti del sito stesso.

Qualunque sia il motivo, abbiamo la necessità di proteggere l'accesso a una pagina. Vediamo adesso un paio di esempi, rimandando poi l'approfondimento del discorso a dopo l'introduzione dei database (*lezione 4*).

PHP ci consente di ottenere la protezione di una pagina in molti modi. Quello scelto per esercizio implica la creazione di una funzione e l'uso di un ciclo di controllo (*listato_08*). La pagina di esempio è *protetta_1.php*, e al primo accesso ci fa vedere un form dove l'unico campo da compilare è quello relativo alla password da inserire.

La funzione controllo si occupa solo di restituire un valore *true* o *false*, a seconda che il confronto tra le due variabili dia esito positivo o negativo. Nel nostro caso, il valore restituito dalla funzione sarà assegnato a una variabile *$check*, la quale chiama la funzione usando come campi la password inserita nel form e la password definita da noi (*pluto*). Se *$check* avrà valore *true* allora la pagina sarà visualizzata, altrimenti ci troveremo di nuovo di fronte al form vuoto.

```
<?php

function controllo ($confronto, $pass) {
if ($confronto==$pass) {
return true;
}
return false;
}

$password=”pluto”;
// la password “pluto” l'abbiamo
impostata noi

$check=controllo($_POST['pwd'],
$password);
// $check assumerà valore true o
false in dipendenza del valore inserito
nel form

if ($check==false) {
// la password è errata o non è mai
stata inserita
?>
<h3>Inserisci la password corretta per
entrare nella pagina protetta</h3>
<form action="protetta_1.php"
method="post">
<input type="password"
name="pwd"> = password<p></p>
<input type="submit" name="invio"
value="Connetti">
</form>
<?php
}
else {
// la password è esatta e quindi
posso avere accesso alla pagina
  echo "<h3>Hai inserito la password
giusta. La pagina è a tua
disposizione</h3>";
}
?>
```

listato 8

Una variazione sullo stesso tema (meno elegante, ma ugualmente efficace) la potete vedere in *protetta_1_var.php*.

Per migliorare la costruzione presentata, bisogna quantomeno posizionare in un file esterno la funzione e il valore assegnato da noi alla password (vedere *password.inc.php* e *protetta_1_inc.php*).

In questo modo è possibile usare il costrutto *include* per inserire senza errori la stessa password in più pagine e se si modifica la password lo si fa in un unico punto.