PHP Introduzione

Prof. Viglietti Francesco
Www.in4matika.altervista.org

PHP: Hypertext Preprocessor

PHP nasce nel 1994 la cui funzione era quella di facilitare ai programmatori l'amministrazione delle homepage personali: da qui trae origine il suo nome, che allora significava appunto Personal Home Page. Oggi PHP è conosciuto come PHP: Hypertext Preprocessor, ed è un linguaggio completo di scripting, sofisticato e flessibile, che può girare praticamente su qualsiasi server Web, su qualsiasi sistema operativo (Windows o Unix/Linux, ma anche Mac, AS/400, Novell, OS/2, ...), e consente di interagire praticamente con qualsiasi tipo di database (MySql, PostgreSql, Sql Server, Oracle, SyBase, Access, ...). Si può utilizzare per i più svariati tipi di progetti, dalla semplice homepage dinamica fino al grande portale o al sito di e-commerce.

PHP e HTML

PHP è un linguaggio la cui funzione fondamentale è quella di produrre codice HTML, che è quello dal quale sono formate le pagine web. Essendo PHP un linguaggio di programmazione, abbiamo la possibilità di analizzare diverse situazioni (l'input degli utenti, i dati contenuti in un database) e di decidere, di conseguenza, di produrre codice HTML condizionato ai risultati dell'elaborazione. Questo è, in parole povere, il Web dinamico.
Quando il server riceve una richiesta per una pagina php, la fa analizzare dall'interprete del PHP stesso, il quale restituisce un file contenente solo il codice che deve essere inviato al browser (in linea di massima HTML, ma può esserci anche codice JavaScript, fogli di stile css ...).

PHP e HTML

Come si realizza la "produzione" di codice HTML?
La prima cosa da sapere è come fa PHP (inteso come interprete) a riconoscere il codice php contenuto nel file che sta analizzando. Il codice php infatti deve essere compreso fra appositi tag di apertura e di chiusura, che sono i seguenti:

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

Tutto ciò che è contenuto fra questi tag deve corrispondere alle regole sintattiche del PHP, ed è codice che sarà eseguito dall'interprete e non sarà inviato al browser. Per generare il codice da inviare al browser abbiamo due costrutti del linguaggio, che possiamo considerare equivalenti, che sono: ***print()*** e ***echo()***.

PHP e HTML esempio:

Vediamo un primo esempio:

```
<HTML><HEAD>
<TITLE>Pagina di prova in PHP</TITLE></HEAD>
<BODY>
<?php
print "Buongiorno a tutti!<br>\n";
echo "E\' una bellissima giornata";
?></BODY>
```

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

```
...[]
<BODY>
Buongiorno a tutti!<br>
E' una bellissima giornata
</BODY>
```

Le variabili 1

Le variabili sono componenti fondamentali di qualsiasi linguaggio di programmazione.
In PHP possiamo scegliere il nome delle variabili usando lettere, numeri ed il trattino di sottolineatura, (_). Il primo carattere del nome deve essere però una lettera o un underscore (non un numero). Dobbiamo inoltre ricordarci che il nome delle variabili è case sensitive, di conseguenza, se scriviamo due volte un nome di variabile usando le maiuscole in maniera differente, per PHP si tratterà di due variabili distinte! Nello script PHP il nome delle variabili è preceduto dal simbolo del dollaro (**$**).
PHP ha una caratteristica che lo rende molto più flessibile rispetto ad altri linguaggi di programmazione: non richiede, infatti, che le variabili vengano dichiarate prima del loro uso. Possiamo quindi permetterci di riferirci ad una variabile direttamente con la sua valorizzazione:

Le variabili 2

```
$a = 9;
$b = 4;
$c = $a * $b;
```

Con questo codice abbiamo valorizzato tre variabili: ad **a**, abbiamo dato il valore 9; a **b**, abbiamo assegnato il valore 4; e infine **c**, che dovrà assumere il valore del prodotto di **a** e **b**. Evidentemente, dopo l'esecuzione del codice **c** varrà 36.

Sopra abbiamo visto l'inizializzazione delle variabili. In realtà, possiamo riferirci ad una variabile anche senza che sia stata inizializzata. Ad esempio, supponendo che nel nostro script non sia stata valorizzata nessuna variabile 'z', potremmo avere un'istruzione di questo genere:

```
print $c;
```

Il risultato sarà che verra stampato il valore di **c** cioè 36

I tipi di variabile 1

Vediamo quali sono i tipi di valore che una variabile può contenere.
Booleano. Le variabili booleane possono avere il loro valore solo TRUE o FALSE.

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

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

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

I tipi di variabile 2

Stringa. è un qualsiasi insieme di caratteri, senza limitazione. Le stringhe possono essere espresse in tre maniere:
Delimitate da apici (‘stringa’)o da virgolette (“stringa”) o con la sintassi heredoc.
Le stringhe delimitate da apici sono la forma più semplice, consigliata quando all'interno della stringa non vi sono variabili di cui vogliamo ricavare il valore:

$nome='Anna';
print "$nome è simpatica... a pochi"; /*stampa: Anna è simpatica... a pochi*/
print '$nome è simpatica... a pochi'; /*stampa: $nome è simpatica... a pochi*/

L'uso delle virgolette permette di risolvere le variabili, mentre con gli apici i nomi di variabile rimangono... invariati.

Ci sono delle regole da ricordare quando si usano le stringhe delimitate da apici o virgolette: può capitare che una stringa debba contenere a sua volta un apice o un paio di virgolette, abbiamo bisogno di far capire a PHP che quel carattere fa parte della stringa e non è il suo delimitatore. In questo caso si usa il cosiddetto 'carattere di escape', cioè la barra rovesciata (backslash: \).

Sintassi heredoc

In questo caso una stringa inizia con **<<<identificatore** e continua finché PHP non incontra una riga di testo di input che consiste solo **nello stesso identificatore allineato a sinistra e ; senza spazi prima o dopo il (;)**.

La notazione heredoc è sicuramente l'ideale per la stesura di grossi blocchi di codice html o javascript (da inviare con un unico comando 'echo') pur mantenendo un certo ordine di codifica. Ad esempio:

```
<?php
echo <<<HTML
    <p>
        Lorem Ipsum is simply dummy text of the printing and pesetting industry.
        Lorem Ipsum has been the industry's standard dummy text ever since
        the 1500s, when an unknown printer took a galley of type and scrambled
        it to make a type specimen book.
    </p>
HTML;
?>
```

Gli array

Possiamo considerare un array come una variabile complessa, che contiene una serie di valori, ciascuno dei quali caratterizzato da una chiave, o indice.

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

A questo punto ciascuno dei nostri cinque colori è caratterizzato da un indice numerico, che PHP assegna automaticamente a partire da 0. L'indice viene indicato fra parentesi quadre dietro al nome dell'array:

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

Gli array

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

$colori[] = 'blu';

Contare gli elementi di un array. Se vogliamo sapere di quanti elementi è composto un array, possiamo utilizzare la funzione **count()**:

$numero_elementi = count($colori);

Concludiamo questa lezione sugli array vedendo, come già abbiamo fatto per le variabili, in che modo possiamo eliminare un intero array o soltanto un suo elemento.

unset($colori[2]); // elimina l'elemento 'giallo'

unset($colori); // elimina l'intero array $colori

Gli operatori

Gli operatori di effettuare le tradizionali operazioni aritmetiche, e di manipolare il contenuto delle nostre variabili. Il più conosciuto è quello di assegnazione:

$nome = 'Giorgio';
$a = 3 + 7; //addizione
$b = 5 - 2; //sottrazione
$c = 9 * 6; //moltiplicazione
$d = 8 / 2; //divisione
$e = 7 % 4; /*modulo (resto della divisione intera, in questo caso 3)*/
$a++; /*incrementa di 1: equivale ad $a = $a + 1, o $a += 1*/
++$a; /*stessa cosa, ma con una differenza nella valutazione dell'espressione */
$a--; //decrementa di 1: equivale ad $a = $a - 1, o $a -= 1
--$a; //anche qui stessa cosa, con la stessa differenza di cui sopra
Concatenare delle stringhe: il punto **'.'** .
$nome = 'pippo';
**$stringa1 = 'ciao ' . $nome;** //$stringa1 vale 'ciao pippo'

Gli operatori di confronto e logici

Gli O. di confronto permettono i confronti fra valori. Confrontando i due valori posti a sinistra e a destra dell'operatore stesso. Il risultato di questa operazione potrà essere, o vero o falso. Questi operatori sono:
== (uguale); != (diverso)
=== : identico (cioè uguale e dello stesso tipo: ad es. due variabili di tipo intero)
> (maggiore); >= (maggiore uguale); < (minore); <= (minore uguale)
Gli o. logici combinano più valori booleani, oppure ne negano uno (NOT):
or: valuta se almeno uno dei due operatori è vero; si può indicare con 'Or' oppure '||'
and: valuta se entrambi gli operatori sono veri; si indica con 'And' o '**&&**'
xor: or esclusivo, valuta se uno solo dei due operatori è vero: l'altro deve essere falso; si indica con 'Xor'
not: vale come negazione e si usa con un solo operatore: in pratica è vero quando l'operatore è falso, e viceversa; si indica con '**!**'

Strutture di controllo if..

if(condizione)
istruzione1
else
istruzione2

Potremmo anche avere la necessità di eseguire, non una sola ma più istruzioni. Questo è perfettamente possibile, ma dobbiamo ricordarci di comprendere questo blocco di istruzioni fra due parentesi graffe, ad esempio così:

```
if ($nome == 'Luca') {
  print "ciao Luca!<br>";
  print "dove sono i tuoi amici?<br>";
}
else {
  print "ciao $nome!";
}
```

Strutture di controllo if..elseif e switch

Un'ulteriore possibilità che ci fornisce l'if è quella di utilizzare la parola chiave 'elseif, con cui possiamo indicare una seconda condizione, da valutare solo nel caso in cui la prima sia falsa:

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

Passiamo ora a verificare una seconda istruzione che ci permette di prevedere diversi valori possibili per un'espressione (switch):

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

Strutture di controllo: for

for(IndiceIniziale;IndiceFinale;Incremento){

…}

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

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

Strutture di controllo: while

Vediamo ora il ciclo 'while', con l'esempio sotto indicato si otteniene lo stesso risultato dell'esempio precedente (slide precedente). Esiste anche un'altra forma, con la quale possiamo assicurarci che il codice indicato tra le parentesi graffe venga eseguito almeno una volta: si tratta del 'do...while'

```
$mul = 1;
while ($mul <= 10) {
  $ris = 5 * $mul;
  print("5 * $mul = $ris<br>");
  $mul++;
}
```

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

Configurazione di PHP: php.ini

Il file php.ini permette di modificare le impostazioni di PHP. Dove si trova questo file? Lo possiamo localizzare, creando il seguente file php:

```
<?php
phpinfo();
?>
```

Salviamo il codice precedente nel file phpinfo.php, richiamandolo dal server otterremo una lunga lista di parametri che raccontano tutto il setup di PHP. In particolare, in una delle prime righe troveremo la voce '**Configuration File (php.ini) Path**', di fianco alla quale potremo vedere dove si trova il 'nostro' php.ini.
Il file è 'voluminoso', perché contiene spiegazioni su ogni impostazione.
Tutte le righe che iniziano con un ; vengono considerate commenti.

Configurazione di PHP 1

short_open_tag: permette, di aprire e chiudere PHP con i tag brevi <? e ?>. Se lo impostiamo a 'off', questi tag non saranno più riconosciuti. (default 'on').
asp_tags: se 'on', permette l'utilizzo di tag in stile ASP <% e %>. (default 'off').
max_execution_time: è il tempo limite concesso a PHP per l'esecuzione di uno script. E' espresso in secondi, (default 30sec). Normalmente non c'è motivo di modificarlo, perché 30 secondi sono un tempo più che sufficiente per eseguire uno script PHP.
error_reporting: rappresenta il 'livello' di errori che vengono mostrati da PHP, (consiglio impostazione 'E_ALL', che mostra tutti gli errori, compresi quelli meno gravi di tipo 'notice'.
display_errors: stabilisce se gli errori devono essere mostrati sul browser. impostandolo ad 'off', non vedremo più gli errori sul browser e, per trovarli, dovremo abilitare la loro registrazione su un file di log.

Configurazione di PHP 2

log_errors / error_log: si usano nel caso in cui, non volessimo mostrare gli errori sul browser. In questo caso dovremo impostare log_errors ad 'on' ed indicare in error_log il nome del file in cui registrare gli errori (con percorso assoluto sul server, ad esempio 'C:\Php\errori.txt'). Il file dev'essere creato prima, altrimenti PHP non riuscirà a memorizzare i suoi errori! E' anche importante che il web server abbia il permesso per scrivere su questo file.

register_globals: (default 'off'). Tale impostazione non ci permette di utilizzare come variabili globali quelle che arrivano dalla query string o dai moduli inviati dagli utenti.

session.save_path: dobbiamo indicare la cartella nella quale PHP andrà a salvare i files di sessione: di default vi è impostato il valore '/tmp', ma possiamo indicare quello che ci fa più comodo (ad es. 'C:\Php\sessioni').

Configurazione di PHP 3

extension_dir: questo parametro va impostato se vogliamo utilizzare le estensioni di PHP, ad es. le librerie gd per lavorare con le immagini, o le librerie pdf per I documenti pdf, o ancora quando vogliamo collegarci a database diversi da MySql, come PostgresSql, Oracle, Sql Server ecc. Tali funzioni richiedono infatti la presenza di apposite librerie (in Windows files .dll che vengono distribuiti insieme al pacchetto PHP e si trovano nella cartella 'extensions'), ed in questo parametro dobbiamo indicare il percorso sulla nostra macchina della directory in cui le abbiamo sistemate.
extension: questo parametro è collegato al precedente, serve per indicare quali sono le estensioni che PHP deve caricare. Dovremo valorizzarne uno per ogni estensione che ci interessa. Il file php.ini possiede già una lista precompilata di queste estensioni, che però sono tutte 'commentate': ci basterà quindi togliere il ; da quelle che vogliamo caricare.

Le principali funzioni di PHP

Una funzione è un insieme di istruzioni che hanno lo scopo di eseguire determinate operazioni, che ci consentono di non dover riscrivere tutto il codice ogni volta che abbiamo la necessità di eseguire quelle operazioni. Basta infatti richiamare l'apposita funzione, fornendole i parametri, cioè i dati, di cui ha bisogno per la sua esecuzione.

Le funzioni possono essere *incorporate* nel linguaggio, oppure essere definite dall'utente. In entrambi i casi, il modo di richiamarle è lo stesso. La sintassi fondamentale con la quale si richiama una funzione, cioé si chiede a PHP di eseguirla, è molto semplice:

nome_funzione();

Si tratta semplicemente di indicare il nome della funzione, seguito da parentesi tonde. Queste parentesi obbligatorie possono contenere i parametri da passare alla funzione.

Funzioni sulle variabili

empty(valore): verifica se qualsiasi variabile passata è vuota.
isset(valore): verifica se la variabile è definita, (cioè se è inizializzata o nun è NULL).
is_null(valore): verifica se la variabile equivale a NULL, ma genera un errore 'notice' se eseguito su una variabile non definita.
is_int(valore), is_integer(valore), is_long(valore): verificano se la variabile è di tipo intero.
is_float(valore), is_double(valore), is_real(valore): verificano se la variabile è di tipo reale.
is_string(valore), is_array(valore) la prima verifica se la variabile è una stringa, l'altra se è un array.
is_numeric(valore): verifica se la variabile contiene un valore numerico. (la differenza fra questa funzione e is_int() o is_float(), é che, queste ultime, nel caso di una stringa che contiene valori numerici, restituiscono falso, mentre is_numeric() restituisce vero.

Funzioni sulle variabili

Tutte le funzioni viste fin ora restituiscono un booleano.

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

print_r(valore): stampa informazioni relative al contenuto della variabile che le abbiamo passato. E' utile in fase di debug, Restituisce un booleano.

unset(valore): distrugge la variabile specificata. Dopo l'unset(), l'esecuzione di empty() o is_null() sulla stessa variabile restituirà vero, mentre isset() restituirà falso. Non restituisce valori.

Esempi:

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

Ancora esempi su Funzioni

```
$b = is_int($a); //falso: $a ora è una stringa
$b = is_string($a); //vero
$b = is_numeric($a); //vero: la stringa ha un contenuto numerico
$c = gettype($b); //$c prende il valore 'boolean'
unset($a); //eliminiamo la variabile $a;
 $b = is_null($a); //vero, ma genera errore
 if (empty($a)) {
        print ('$a è vuota o non definita!');
        } else {
        print ('$a contiene un valore');
        }
if (is_numeric($a)) {
  print ('$a contiene un valore numerico');
} else {
  print ('$a non contiene un numero');
}
```

Funzioni sulle stringhe 1

strlen(stringa): verifica la lunghezza della stringa. Restituisce un numero intero.
trim(stringa): elimina gli spazi all'inizio e alla fine della stringa.
ltrim(stringa): li elimina all'inizio della stringa.
rtrim(stringa): li elimina alla fine della stringa.
Le tre funzioni trim, restituiscono la stringa modificata.
substr(stringa, intero [, intero]): restituisce una porzione della stringa, in base al secondo parametro (che indica l'inizio della porzione da estrarre), e all'eventuale terzo parametro, che indica quanti caratteri devono essere estratti. Da notare che i caratteri vanno contati a partire da zero. Si può anche indicare un numero negativo come carattere iniziale: in questo caso, il carattere iniziale della porzione di stringa restituita verrà contato a partire dal fondo. Es., con substr(stringa, -5, 3) si otterranno tre caratteri a partire dal quintultimo.

Funzioni sulle stringhe 2

str_replace(stringa, stringa, stringa): effettua una sostituzione della prima stringa con la seconda all'interno della terza. Ad esempio: str_replace('p', 't', 'pippo') sostituisce le 'p' con le 't' all'interno di 'pippo', e quindi restituisce 'titto'. Restituisce la terza stringa modificata.

strpos(stringa, stringa): cerca la posizione della seconda stringa all'interno della prima. Ad esempio: strpos('Lorenzo', 're') restituisce 2, ad indicare la terza posizione. Restituisce un intero che rappresenta la posizione a partire da 0 della stringa cercata. Se la seconda stringa non è presente, restituisce FALSE.

stripos() fa la stessa ricerca senza differenza fra maiuscole e minuscole.

strstr(stringa, stringa): cerca la seconda stringa all'interno della prima, e restituisce la prima a partire dal punto in cui ha trovato la seconda. strstr('Lorenzo', 're') restituisce 'renzo'. Restituisce una stringa se la ricerca va a buon fine, altrimenti FALSE.

stristr(): funziona allo stesso modo senza differenza fra maiuscole e minuscole.

Funzioni sulle stringhe 3

strtolower(stringa): converte tutti i caratteri alfabetici nelle corrispondenti lettere minuscole.
strtoupper(stringa): li converte nelle corrispondenti lettere maiuscole.
ucfirst(stringa): trasforma in maiuscolo il primo carattere della stringa.
ucwords(stringa): trasforma in maiuscolo il primo carattere di ogni parola della stringa, intendendo come parola una serie di caratteri che segue uno spazio.
Le quattro funzioni precedenti, restituiscono la stringa modificata.
explode(stringa, stringa [, intero]): trasforma la seconda stringa in un array, usando la prima per separare gli elementi. Il terzo parametro può servire ad indicare il numero massimo di elementi che l'array può contenere. Ad esempio: explode(' ', 'ciao Mario') restituisce un array di due elementi in cui il primo è 'ciao' e il secondo 'Mario'. Restituisce un array.

Esempi su Funzioni sulle stringhe

$a = 'IERI ERA DOMENICA';
$b = strtolower($a);
// $b diventa 'ieri era domenica', $a rimane 'IERI ERA DOMENICA'
strlen('abcd'); //restituisce 4
trim(' Buongiorno a tutti   '); //restituisce 'Buongiorno a tutti'
substr('Buongiorno a tutti', 4); // 'giorno a tutti' (inizia dal quinto)
substr('Buongiorno a tutti', 4, 6); // 'giorno'(6 caratteri a partire dal 5°)
substr('Buongiorno a tutti', -4); // 'utti' (ultimi quattro)
substr('Buongiorno a tutti', -4, 2); // 'ut' (2 caratteri a partire dal quartultimo)
substr('Buongiorno a tutti, 4, -2); // 'giorno a tut' (dal quinto al terzultimo)
str_replace('Buongiorno', 'Ciao', 'Buongiorno a tutti'); // 'Ciao a tutti'

Esempi su Funzioni sulle stringhe

str_replace('dom', 'x', 'Domani è domenica'); // 'Domani è xenica'
str_ireplace('dom, 'x', 'Domani è domenica'); // 'xani è xenica'
strpos('Domani è domenica', 'm'); // 2 (prima 'm' trovata)
strstr('Domani è domenica', 'm'); // 'mani è domenica' (dalla prima 'm')
strtoupper('Buongiorno a tutti'); // 'BUONGIORNO A TUTTI'
ucfirst('buongiorno a tutti'); // 'Buongiorno a tutti';
ucwords('buongiorno a tutti'); // 'Buongiorno A Tutti';
explode(',','Alberto,Mario,Giovanni'); /* suddivide la stringa in un array, separando un elemento ogni volta che trova una virgola; avremo quindi un array di tre elementi: ('Alberto','Mario','Giovanni')*/
explode(',','Alberto,Mario,Giovanni',2); /* in questo caso l'array può contenere al massimo due elementi, per cui nel primo elemento andrà 'Alberto' e nel secondo il resto della stringa: 'Mario,Giovanni' */

Funzioni sugli array 1

count(array): conta il numero di elementi dell'array. Restituisce un intero.
array_reverse(array [, booleano]): inverte l'ordine degli elementi dell'array. Per tenere le chiavi si deve passare TRUE nel II parametro . Restituisce l'array con gli elementi invertiti.
sort(array): Modifica direttamente l'array passato in input. I valori vengono disposti in ordine crescente. ***rsort(array)***: ordina in ordine decrescente. ***asort(array)***: come sort(), ma vengono mantenute le chiavi originarie. ***arsort(array)***: come rsort(), ma mantiene le chiavi originarie. Non restituiscono nulla.
in_array(valore, array): cerca il valore all'interno dell'array. Restituisce un valore booleano.
array_key_exists(valore, array): cerca il valore fra le chiavi dell'array. Restituisce boolean.

Funzioni sugli array 2

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

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

array_pop(array): estrae l'ultimo elemento dell'array, che viene 'accorciato'. Restituisce l'elemento in fondo all'array e, modifica l'array in input togliendogli lo stesso elemento.

Funzioni sugli array 3

array_push(array, valore [,valore...]): accoda i valori indicati all'array. Equivale all'uso dell'istruzione di accodamento $array[]=$valore , ma permette di accodare più valori contemporaneamente. Restituisce il numero degli elementi dell'array dopo l'accodamento.

array_shift(array): estrae il primo elemento dell'array .Anche in questo caso l'array viene 'accorciato', ed inoltre gli indici numerici vengono rinumerati. Rimangono invece invariati quelli associativi. Restituisce l'elemento estratto dall'array.

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

implode(stringa, array): opposta di explode(), riunisce in un'unica stringa i valori dell'array. Restituisce la stringa risultato dell'aggregazione. Sinonimo è ***join()***.

Funzioni su date e ore

time(): fornisce il timestamp relativo al momento in cui viene eseguita. Restituisce un int.

date(formato [,timestamp]): considera il timestamp in input (se non è indicato, prende quello attuale) e fornisce una data formattata secondo le specifiche indicate nel primo parametro.

date(): restituisce la stringa formattata che rappresenta la data.

mktime(ore, minuti, secondi, mese, giorno, anno): calcola il timestamp, ma possiamo utilizzarla per fare calcoli sulle date. Infatti, se ad es. nel parametro mese passiamo 14, PHP lo interpreterà come 12+2, cioè "febbraio dell'anno successivo", e quindi considererà il mese come febbraio ed aumenterà l'anno di 1. Restituisce un int.

checkdate(mese, giorno, anno): verifica se i valori passati costituiscono una data valida. Restituisce un valore booleano.

Funzioni su date e ore

Tali specifiche si basano su una tabella di cui riassumiamo i valori più usati:
Y - anno su 4 cifre
y - anno su 2 cifre
n - mese numerico (1-12)
m - mese numerico su 2 cifre (01-12)
F - mese testuale ('January' - 'December')
M - mese testuale su 3 lettere ('Jan' - 'Dec')
d - giorno del mese su due cifre (01-31)
j - giorno del mese (1-31)
w - giorno settimana, numerico (0=dom, 6=sab)
l - giorno settimana, testuale ('Sunday' - 'Saturday')
D - giorno settimana su 3 lettere ('Sun' - 'Sat')
H - ora su due cifre (00-23)
G - ora (0-23)
i - minuti su due cifre (00-59)
s - secondi su due cifre (00-59)

Funzioni personalizzate 1

Vediamo come definire una funzione, fermo restando che, al momento di eseguirla, la chiamata si svolge con le stesse modalità con cui vengono chiamate le funzioni incorporate del linguaggio. Immaginiamo, di voler costruire una funzione che, dati tre numeri, ci restituisca il maggiore dei tre:

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

Funzioni personalizzate 2

la definizione della funzione avviene attraverso la parola chiave function, seguita dal nome che abbiamo individuato per la funzione, e dalle parentesi che contengono i parametri che devono essere passati alla funzione. Di seguito, contenuto fra parentesi graffe, ci sarà il codice che viene eseguito ogni volta che la funzione viene richiamata.
All'interno della funzione l'istruzione return...; segna il termine della funzione, cioè restituisce il controllo allo script nel punto in cui la funzione è stata chiamata, e contemporaneamente determina anche il valore restituito dalla funzione. Nel nostro esempio, i tre dati ricevuti in input vengono controllati, uno dopo l'altro, per verificare che siano numerici: in caso negativo (il test infatti viene fatto facendo precedere la funzione is_numeric() dal simbolo ! di negazione), la funzione termina immediatamente, restituendo il valore booleano FALSE.