Authors and contributors
Fabio Proietti

Feedback
Please direct any comments or suggestions about this document to
fabio.proietti AT istruzione DOT it

Publication date and version
2011-09-17, v.0.3

last modified 2014-09-30

PHP 5

PHP significa PHP Hypertext Preprocessor ed è un linguaggio interpretato orientato agli oggetti.
Il seguente manuale è tratto dal sito: `http://www.php.net`

> Suggerimento: prima di procedere ulteriormente nello studio, ripassare la differenza tra linguaggio interpretato e compilato.

Le applicazioni più comuni dei programmi scritti in linguaggio PHP vengono eseguite all'interno di un web server.

> Suggerimento: prima di procedere ulteriormente nello studio si consiglia di ripassare il modello di rete "client/server" e il concetto di programmazione lato client e lato server.

PHP è un linguaggio che viene integrato all'interno del codice HTML o XHTML.

> Suggerimento: prima di procedere ulteriormente nello studio, ripassare la differenza tra linguaggio HTML e XHTML.

Come il codice HTML, anche il codice PHP si scrive all'interno di un file di testo non formattato (come un file.txt) usando di solito l'estensione .php.
Le istruzioni PHP devono essere racchiuse tra i seguenti due simboli che indicano l'inizio e la fine di un codice:

```
<?php

?>
```

Esempio di pagina "prova.php"

```
<!DOCTYPE html >

<html>
   <head>
      <title> La mia pagina
      </title>
      <meta name="author" content="Mario Rossi" >
      <meta charset="utf-8" >
   </head>
   <body>
      <h1>Questo &egrave; un titolo</h1>

      <?php
         echo '<h2>Questo &egrave; un test in PHP</h2>'
      ?>

   </body>
</html>
```

Quando un browser (un client) richiede al server web la visualizzazione della pagina "prova.php", cosa accade?
Prima di tutto il server web interpreta il codice PHP che si trova nella pagina, nel precedente esempio è evidenziato in giallo. In questo esempio il codice chiede di scrivere (usando echo) nella pagina web il testo racchiuso dagli apici.

Il blocco del codice PHP viene quindi sostituito solo dal codice di output, scritto tra apici, e la nuova pagina "prova.php" così ottenuta, ovvero senza il codice PHP, viene inviata al client (al browser) che la mostra all'utente. Il browser non sarebbe stato in grado di interpretare le istruzioni in PHP...

Esempio, la pagina prova.php, così come la riceve il browser

```
<!DOCTYPE html>

<html>
   <head>
      <title> La mia pagina
      </title>
      <meta name="author" content="Mario Rossi" >
      <meta  charset="utf-8" >
   </head>
   <body>
      <h1>Questo &egrave; un titolo</h1>

<h2>Questo &egrave; un test in PHP</h2>

   </body>
</html>
```

Le istruzioni nel codice PHP non ci sono più perché sono state eseguite dall'interprete di linguaggio del server web, per questo motivo questo sistema viene chiamato anche programmazione *lato server* o scripting *server side*. Viceversa, se fosse stato presente un codice in linguaggio Javascript, questo sarebbe stato inviato direttamente al browser ed eseguito dal *lato client*.

Licenza d'uso

La licenza sotto cui viene pubblicato il linguaggio PHP è una licenza che permette di usarlo e modificarlo gratuitamente. Questo tipo di licenza è considerata una licenza open source, ma non è considerata compatibile con la famosa licenza GPL per le restrizioni sull'utilizzo di alcuni termini.
http://www.php.net/license/3_01.txt

```
--------------------------------------------------------------------
                  The PHP License, version 3.01
Copyright (c) 1999 - 2010 The PHP Group. All rights reserved.
--------------------------------------------------------------------
--------------------------------------------------------------------
This software consists of voluntary contributions made by many
individuals on behalf of the PHP Group. The PHP Group can be contacted via Email
at group@php.net. For more information on the PHP Group and the PHP project,
please see <http://www.php.net>.
```

Separare le istruzioni

Come nel linguaggio C, anche nel PHP tutte le istruzioni di un blocco, devono essere terminate dal punto e virgola. Non occorre inserire il punto e virgola per chiudere l'ultima riga di un blocco PHP.

Esempio di blocco costituito da una sola riga (che è anche l'ultima riga)

```
<?php
        echo "<h1>Questo &egrave; un test</h1>";
?>
```

```
<?php
        echo "<h1>Questo &egrave; un test</h1>"
?>
```

Commenti

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

```
<?php
    echo "Questo &egrave; un test"; // un commento su una linea nello stile c++

                            /* Questo è un commento
                            su più linee */

    echo "Questo &egrave; un altro test"; # un commento stile shell Unix
?>
```

Tipi di dato

PHP prevede otto tipi primitivi, costituiti da:
Quattro tipi semplici (o scalari):

- `boolean (o bool)`
- `integer (o int)`
- `float`
- `string`

Due tipi strutturati:

- `array`
- `object`

E due tipi speciali:

- `resource`
- `NULL`

Il tipo di una variabile non è sempre impostato dal programmatore, ma può essere anche deciso al momento dell'esecuzione del programma (runtime)

dall'interprete del PHP, a seconda del contesto in cui la variabile è usata.
Questa caratteristica colloca il linguaggio PHP tra quelli cosiddetti "debolmente tipizzati".
Seguono alcuni esempi di assegnazione, in cui la variabile viene inizializzata con un valore:

```
$var1=90;/         // la variabile è inizializzata con un valore (diventa
                   // automaticamente di tipo int)
$var2="100";       // la variabile è inizializzata (diventa
                   // automaticamente di tipo string)
settype($var2, "int");      // la variabile è CONVERTITA integer
$var3="ciao";      // la variabile è automaticamente di tipo string
$var4;             // questa e' dichiarata ma non e' inizializzata
```

Se una variabile viene dichiarata ma non viene inizializzata con nessun valore, allora le viene automaticamente assegnato il tipo di dato speciale NULL. Le variabili di questo tipo possono assumere solo il valore della costante NULL.
Non confondere il nome del tipo NULL con il nome della costante NULL.
Sarebbe come confondere il tipo di dato integer con la costante 90.

Una variabile di tipo resource viene usata per contenere dati speciali come le connessioni ai database o collegamenti a file aperti in lettura.

Variabili

Le variabili in un linguaggio di programmazione sono delle aree della memoria (paragonabili a dei cassetti) in cui il programmatore può memorizzare dei valori (paragonabile al contenuto dei cassetti). Le variabili in PHP (i cassetti) sono identificate da un simbolo del dollaro $ seguito dal nome della variabile. I nomi delle variabili sono sensibili al maiuscolo/minuscolo.
Un nome valido di una variabile non deve iniziare con un numero, ma deve iniziare con una lettera o il trattino basso (underscore), seguito da un numero qualsiasi di lettere, numeri o underscore.

> **Nota**: *$this* è una variabile speciale per la programmazione orientata agli oggetti.

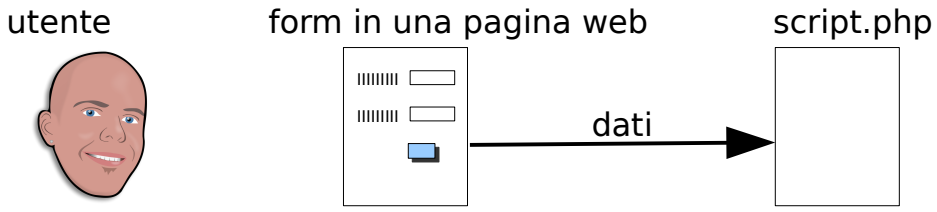
```
<php
$var = "Mario";
$Var = "Anna";
echo $var;      // visualizza Mario
echo $Var;      // visualizza Anna

$4site = "no";     // sbagliato perche` inizia con un numero
$p4site = "ok";    // corretto
?>
```

Promemoria: qui c'è una sezione nascosta che contiene un approfondimento sul passaggio per valore e per variabile. In PHP non è necessario inizializzare variabili, ma resta una buona pratica farlo. Le variabili non inizializzate possono valere ZERO, FALSE oppure la stringa vuota, a seconda del tipo.

Form (moduli)

Si è visto che le variabili possono essere dichiarate dal programmatore nella pagina php oppure utilizzate senza essere esplicitamente dichiarate, mediante inizializzazione.
Si vedranno due tipi di esempi: il primo in cui i valori con cui vengono inizializzate le variabili, possono essere inviati da un utente tramite un form e il secondo in cui le variabili possono essere inizializzate dal programmatore nella stessa pagina php.
Nel primo caso i nomi delle variabili sono definiti anche nel form da cui sono state inviate.



Esempio di frammento di codice della pagina statica dove viene chiesto nome ed età di una persona:

```
<form action="action.php" method="post">
 <p>Il tuo nome: <input type="text" name="var_nome" ></p>
 <p>La tua et&agrave;: <input type="text" name="var_eta" ></p>
 <p><input type="submit" ></p>
</form>
```

Esempio di frammento di codice della pagina php che riceve i dati e li visualizza senza elaborarli:

```
<p>
Ciao <?php echo $_POST['var_nome']; ?>.
</p>
<p>
Tu hai <?php echo $_POST['var_eta']; ?> anni.
</p>
```

La pagina dinamica mostrerà qualcosa come:

```
Ciao Mario.

Tu hai 25 anni.
```

Concatenazione

Il termine "concatenazione" significa unire due o più variabili string, una di seguito all'altra, per formare una string più lunga. L'operatore di concatenazione è il punto.
Quello che segue è un esempio in cui le variabili sono inizializzate dal programmatore, nella pagina .php.
Esempio:

```
$var1="Scuo";
$var2="la";
$var3 = $var1.$var2;
echo "<p>questa parola $var3 &egrave; una concatenazione</p>";
echo "<p>questa parola". $var3 ."&egrave; una concatenazione</p>";
// le ultime due fanno la stessa cosa, ma attenzione ai colori...
```

String

Una `string` è una serie di caratteri. Nelle vecchie versioni di PHP, ogni carattere corrisponde ad un Byte (8 bit) e può quindi assumere solo i 255 valori della vecchia codifica ASCII.

> **Nota**: una string può diventare anche molto lunga. L'unico limite è la dimensione della memoria del computer dove viene eseguito lo script PHP.

Apice o apostrofo (single quote) e Virgolette (double quotes)

Come si è già visto in alcuni esempi, il valore contenuto in una variabile string può essere rappresentato racchiuso tra apici ('...'). All'interno degli apici le variabili non vengono espanse.
I caratteri speciali come l'apice e il carattere backslash (\) devono essere rappresentati in una stringa come una sequenza di due caratteri (escaped sequence):

echo ' \' '; stampa l'apice nel codice HTML
echo ' \\ '; stampa il backslash nel codice HTML

Il valore contenuto in una variabile string può essere rappresentato racchiuso anche tra doppie virgolette ("..."). All'interno delle virgolette le variabili vengono espanse e possono essere rappresentati ulteriori caratteri speciali con backslash.

echo " ' "; inserisce l'apice nel codice HTML
echo " \n "; inserisce un newline (a capo) nel codice HTML
echo " \$ "; inserisce il carattere dollaro nel codice HTML

```
<?php

echo 'una semplice stringa';

echo 'si pu&ograve; anche
andare a capo';
```

```
// visualizza: il termine "alunno" è un'entità
echo 'il termine "alunno" è un\'entit&agrave;\n';

// visualizza: le variabili $var1 non vengono espanse
$var1 = 'sole';
echo 'la variabile $var1 non viene espansa tra apici\n';

// visualizza: questo è il contenuto della variabile $var1: sole
echo "questo è il contenuto della variabile \$var1:". $var1;

?>
```

Convertire in string

Il programmatore può convertire una variabile di un tipo, in un altro tipo, usando `settype()` ma di solito le conversioni sono fatte automaticamente nelle espressioni ogni volta che ce n'è bisogno. Per esempio, quando si usano le funzioni `echo()` oppure `print()` con le variabili numeriche, queste vengono convertite in variabili string. In modo simile una variabile boolean TRUE è convertita nella stringa "1", mentre FALSE in "" (la stringa vuota).

Il tipo array è convertito sempre nella stringa "Array", a causa di ciò le funzioni `echo()` e `print()` non possono mostrare il contenuto di un array.

```
$a = Array('primo'=>"Anna"); // si crea un array
echo $a; //cosa stampa?
```

Per visualizzare un singolo elemento di un array si usa un costrutto come questo:

```
echo $a['primo']; //cosa stampa?
```

Un array può contenere molti elementi. Per visualizzare automaticamente tutti gli elementi di un array saranno usati altri costrutti.

Array

In molti linguaggi di programmazione un array è una **variabile strutturata**, cioè è costituita da un insieme di valori (mentre le variabili semplici possono contenere un solo valore). Per poter accedere ai singoli elementi di un array si deve specificare la **posizione** tra **parentesi quadrate** (di solito un numero intero chiamato "chiave"). In molti linguaggi di programmazione un array può contenere solo valori dello stesso tipo, in questo esempio contiene solo numeri interi.

posizione	0	1	2	3
valore	22	101	61	-6

Anche in PHP un array è un tipo di dato in cui possono essere memorizzati più valori e ogni valore è associato ad una chiave, che ne determina la posizione. Ogni elemento di un array può essere a sua volta un array ed è possibile costruire array multidimensionali (saranno fatti solo alcuni esempi in merito).
La chiave è detta anche indice e, oltre ad essere un numero integer, può essere una string. Gli array che usano delle string come chiavi sono detti anche array **associativi**.
In PHP gli elementi di un array non devono essere necessariamente dello stesso tipo, ma possono contenere anche valori di tipi diversi tra loro.

chiave	'nome'	'cognome'	12
valore	"Mario"	"Rossi"	TRUE

Un array può essere creato con il comando `array()`.
Questo costrutto ha come parametri, nelle parentesi, le coppie nella forma :
CHIAVE => VALORE
separati da una virgola.

```
 sintassi tratta dal manuale su php.net
```

```
array(  chiave =>  valore  , ...
     )
```

chiave può essere soltanto integer o string
valore può essere di qualsiasi tipo primitivo del PHP

Esempio:

```
<?php
$a = array(
         'nome' => "Mario",
         'cognome' => "Rossi"
         );

echo "\n<p>".$a['cognome']."</p>\n";  // stampa <p>Rossi</p>
echo "\n<p>"."ciao ". $a['cognome']."</p>\n";

echo "\n<p>".$a."</p>\n"; //cosa stampa echo?
print "\n<p>".$a."</p>\n"; //cosa stampa print?
print_r "\n<pre>\n".$a."\n</pre>\n"; //cosa stampa print_r?
?>
```

Esempio:

```
$b = array(
         'sesso' => "M"
         12 => TRUE
         );

echo "\n<p>".$b['sesso']."</p>\n";    // stampa <p>M</p>
echo "\n<p>".$b[12]."</p>\n";         // stampa TRUE, cioè <p>1</p>
```

Il precedente esempio mostra un array che usa due chiavi di tipo diverso: una chiave di tipo *string* e una *integer*. L'esempio mostra anche che il valore può essere di tipo boolean.

Attenzione alla differenza tra apici (apostrofo) e virgolette (doppie):
nei precedenti esempi, le chiavi string sono racchiuse tra apici semplici, mentre i valori string sono racchiusi tra doppie virgolette.
Questa non è una regola del linguaggio PHP, ma una consuetudine usata spesso dai programmatori. Il seguente esempio mostra diversi modi (alcuni corretti e alcuni sbagliati) in cui può essere visualizzato il contenuto di un array

```
<?php
          $a = array(
                              'primo' => "Mario",
                              'secondo' => "Rossi"
                    );

// diversi modi di eseguire output
          echo "nome: ".$a[primo]; //errore
          echo "nome: ".$a['primo']; //ok
          echo "nome: ".$a["primo"]; //ok

?>
```

Si ricorda che le chiavi possono essere integer o string.
Le chiavi di tipo *Float* sono troncate come un *integer*. Gli array indicizzati con numeri e gli array indicizzati con parole (o “array associativi”), sono in realtà la stessa cosa in PHP, ed entrambi possono avere come indici sia integer che string.
NOTA: una chiave può assumere anche il valore stringa vuota "".

Esempio di array multidimensionale

```
<?php

//esempio di array multidimensionale

$a = array(
         'scatola' => array(
                              66 => 5,
                              13 => 9,
                              'a' => 42
                             )
         );

echo $a['scatola'][66];   // visualizza 5
echo $a['scatola'][13];   // visualizza 9
echo $a['scatola']['a'];  // visualizza 42

?>
```

Nella definizione dell'array può essere indicato il valore, lasciando sottintesa la chiave:

```
<?php

// Questi due vettori sono equivalenti ...
$a1 = array(5 => 43, 32, 56, 'b' => 12);

// ...
$a2 = array(5 => 43, 6 => 32, 7 => 56, 'b' => 12);
?>
```

Avviso: per evitare incompatibilità con le versioni precedenti, è meglio evitare chiavi negative.

Un array (esistente) può essere successivamente modificato mediante un'istruzione di assegnazione, indicando la chiave (l'indice) tra parentesi quadrate:

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

```
<?php

$a = array(        5 => 1,
                   12 => 2
         );

$a['x'] = 42; // questo _aggiunge_ nell'array
              // un elemento con chiave 'x'

$a[5] = 0;    // questo _modifica_ il primo elemento

unset($a[5]); // rimuove l'elemento
unset($a);    // elimina l'intero array

?>
```

Funzioni elementari di output

Come in altri linguaggi, anche in PHP **le funzioni** eseguono un particolare compito su un determinato argomento, che viene passato alla funzione racchiuso tra **parentesi tonde**.

Esempio che usa funzioni:

Si vuole calcolare l'età di una persona a partire dalla sua data di nascita, che si trova nella variabile `$varnascita`, inviata dall'utente tramite un form in HTML.

```
<form method="get" action="script.php">
    data di nascita nel formato YYYY-MM-DD
    <input type="text" name="vargiorno" >
    <input type="submit" value="invia" >
</form>
```

Il server può calcolare e visualizzare l'età eseguendo le istruzioni contenute in una pagina in PHP

```
<?php
   // ricevo la data e tolgo gli spazi bianchi
   $varnascita = trim(htmlentities($_POST['varnascita']));

   //  $varnascita e' una string come "2011-12-31"

   $nascita = strtotime($varnascita);  // trasformo in integer
   $secondi = time()-$nascita;    // e' integer
   $anni= floor($secondi/(365*24*60*60)); //divido e arrotondo

   echo "La tua et&agrave; vale: " . $anni;
?>
```

Al termine dell'esecuzione della funzione, quando il controllo ritorna al programma principale, la funzione può restituire un valore.

Ad esempio, inventando una nuova funzione "scrivi()" (che non esiste in PHP)

nel manuale si troverà scritto

```
int scrivi (string $var)
```

che significa che la funzione accetta come parametro una variabile di tipo string e restituisce un valore intero. In tale esempio il valore intero potrebbe corrispondere al numero delle lettere scritte oppure ad un codice di errore e si userebbe in questo modo

```
scrivi("poche parole");
```

print — Visualizza una stringa

```
int print ( string $argomento )
```

Visualizza il contenuto della variabile *argomento* . Restituisce sempre *1*,

In realtà **print()** non è una funzione (è un costrutto del linguaggio) pertanto le parentesi per la lista degli argomenti non sono obbligatorie.

echo — Visualizza una o più stringhe

```
void echo ( string $arg1 [, string $... ] )
```

Visualizza tutti i parametri.
echo() in realtà non è una funzione (è un costrutto del linguaggio) pertanto non richiede l'uso delle parentesi, infatti, se si vuole passare più di un parametro, non bisogna racchiuderli tra parentesi.

foreach - Esegue un ciclo sugli elementi array

foreach, come i precedenti comandi, usa le parentesi tonde ma non è una funzione. Permette di percorrere gli elementi di un array per mezzo di un puntatore interno che avanza automaticamente. Si tratta di un comando che permette di ripetere automaticamente delle istruzioni, tante volte quanti sono gli elementi che compongono l'array.
foreach funziona solo con array e genera un errore se si tenta di utilizzarlo con variabili di tipo differente. Questa struttura di controllo permette di visualizzare il contenuto di un array, senza dover conoscere la dimensione dell'array o i nomi delle sue chiavi.
Nei seguenti esempi, con *$var* si indica l'array che si vuole visualizzare.
Il seguente esempio attraversa tutto l'array di nome *$var*. Ad ogni ripetizione, si assegna il valore dell'elemento attuale alla variabile temporanea *$val* e il valore della chiave viene assegnato alla variabile *$chi* ed entrambi possono essere visualizzati. Il puntatore interno avanza poi di una posizione (in modo tale che al ciclo successivo l'elemento corrente sarà il successivo elemento dell'array).

```
foreach($var as $chi => $val)
{
         echo "<p>posizione: ". $chi ;  // queste istruzioni sono ripetute
         echo " valore: ". $val."</p>"; // per ogni elemento dell'array
}
```

Nota: All'inizio dell'esecuzione di un ciclo *foreach* il puntatore interno viene automaticamente posizionato nella prima posizione. Questo significa che non è necessario utilizzare la funzione reset() prima di un ciclo *foreach*.

print_r — Stampa il contenuto di una variabile in formato leggibile

```
bool print_r ( mixed $var [, bool $return ] )
```

Questa funzione stampa delle informazioni sul contenuto di una variabile in un formato **facilmente leggibile**.
Se la variabile contiene una stringa, un intero o un numero decimale, viene visualizzato semplicemente il valore stesso.
Se la variabile contiene un array i valori vengono visualizzati in un formato che evidenzia le chiavi ed i relativi elementi. Una notazione simile viene utilizzata per gli oggetti.
Occorre ricordarsi che **print_r()** posiziona il puntatore dell'array alla fine. Pertanto è necessario utilizzare reset() per riportarsi all'inizio.

```
<pre>

   <?php
       $a = array (
         'a' => 'mela',
         'b' => 'banana',
         'c' => array ('x','y','z')
          );
       print_r ($a);
   ?>

</pre>
```

Il precedente esempio, produrrà, all'interno di una pagina web questo codice:

```
<pre>
Array
(
    [a] => mela
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

Se si desidera catturare l'output di **print_r()** in una variabile string, occorre utilizzare il parametro *return* . Se questo viene impostato a **TRUE**, print_r() restituirà l'output per poterlo memorizzare in una variabile, anziché per visualizzarlo (come accade per default).

key()— restituisce la chiave dell'elemento attualmente puntato in un array

```
mixed key(array $var)
```

esempio

```
$x = array (
    ['a'] => "arancia",
    ['b'] => "banana");

$c=key($x);
echo "<p>chiave: ".$c." "valore: .$x[$c]."</p>";

// di solito viene puntato il primo elemento,
// a meno che vengano usate delle funzioni che spostano il puntatore
```

Esempio dell'uso di *return*

```
<?php
    $a = array ('m' => 'monkey', 'foo' => 'bar', 'x' => array ('x', 'y', 'z'));

    $results = print_r($a, true); //$results ora contiene l'output di print_r
?>
```

Esempio

```
<?php
// Create a simple array.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Now delete every item, but leave the array itself intact:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Append an item (note that the new key is 5, instead of 0).
$array[] = 6;
print_r($array);

// Re-index:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Il precedente esempio visualizzerà:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Funzioni PHP per l'uso di un database di SQLite3

http://it2.php.net/manual/en/ref.pdo-sqlite.connection.php

A partire dalla versione 5.3 del linguaggio PHP è possibile utilizzare la notazione della programmazione orientata agli oggetti e l'interfaccia PDO (PHP Data Objects) per l'accesso ai file di SQLite3.

http://it2.php.net/manual/en/book.pdo.php

SQLite3 non è un DBMS server e si può usare solo in un file locale, ovvero, SQLite3 salva ogni database in un file, che deve essere sullo stesso computer dove si trovano le pagine php.

```
//crea un oggetto PDO e lo memorizza nella variabile $db
$db = new PDO("sqlite:/var/www/public/database.sq3");

//esegue un comando che non restituisce nulla
$db->exec("CREATE TABLE tabellina (cf CHAR(16) PRIMARY KEY);");

//esegue una interrogazione che restituisce un oggetto espressione
$espressione = $db->query("SELECT * FROM tabellina;");

//estrae il risultato dall'oggetto espressione
// nota: usare fetchAll() non usare fetch_all()
$risultato = $espressione->fetchAll();

echo "<pre>"; print_r($risultato);echo "</pre>";
```

Funzioni PHP per la comunicazione con un server DBMS

A partire dalla versione 5.3 del linguaggio PHP è possibile utilizzare la notazione della programmazione orientata agli oggetti e l'interfaccia PDO (PHP Data Objects) per la connessione al DBMS PostgreSQL e MySQL.

Si rimanda al manuale per approfondire la sintassi di ogni DBMS

`http://it2.php.net/manual/en/ref.pdo-pgsql.connection.php`

`http://it2.php.net/manual/en/ref.pdo-mysql.connection.php`

```
<?php
  $db = new PDO("pgsql:     host=127.0.0.1;
                            port=5432;
                            dbname=esempio;
                            user=mario;
                            password=pass");
?>
```

Funzione PDO::PDO()

Questo è un esempio di connessione al database MySQL

```
$db= new PDO("mysql:host=127.0.0.1;dbname=registroauto",$user,$password);
```

1 2 3

A seconda delle preferenze del programmatore, questi 3 argomenti, che devono essere separati da 2 virgole, possono essere costanti string racchiuse tra virgolette, oppure variabili.

```
<?php     $dbms="mysql";
          $host="127.0.0.1";
          $dbname="registroauto";
          $user="mario";
          $password="segreto";
          $db= new PDO($dbms.":host=".$host.";dbname=".$dbname,$user,$password);
?>
```

Attenzione: new PDO(...) resituisce un'oggetto memorizzato in $db

Funzione PDO::query()

Ipotizzando di voler eseguire la seguente interrogazione:

```
"SELECT * FROM tabella WHERE nome='Anna';"
```

si può scrivere

```
$nome="Anna";

$espressione = $db->query("SELECT * FROM tabella WHERE nome='".$nome."';");
```

la variabile $db è necessaria per poter eseguire query().

A sua volta viene restituito un oggetto memorizzato in $espressione

Funzione PDOStatement::fetchAll()

Questo metodo viene eseguito dopo query() per ottenere l'insieme di tutte le righe del risultato dell'interrogazione in un array di array

```
$risultato = $espressione->fetchAll(PDO::FETCH_ASSOC);
```

È necessario usare la variabile $espressione.
A sua volta viene restituito un oggetto memorizzato in $risultato

PDO::FETCH_ASSOC è solo una costante numerica predefinita che specifica di usare per le chiavi degli array gli stessi nomi usati nelle tabelle del database (se non si usa questa costante si ottiene il doppio delle righe nel risultato)

$risultato è un array di array e può essere visualizzato con foreach()

Variabili predefinite

Oltre alla variabili create dal programmatore ve ne sono alcune che hanno un nome e una funzione fissa. Si riconoscono dal carattere "underscore" (trattino basso) e dal testo maiuscolo. NOTA: spesso si tratta di variabili superglobali. Significa che non sono locali, ma sono visibili in tutte le parti di uno script.

$_GET

$_GET è una variabile di tipo array che contiene le variabili inviate dalla pagina precedente (ad esempio, da un form) alla pagina attuale, usando il method “get”. Ogni elemento di questo array contiene una delle variabili che è stata inviata.
Nell'ipotesi che si invii, tramite URL: http://indirizzo/pag.php/?name=Anna

```
<?php
         echo 'Ciao ' . $_GET["nomeutente"] . '!';

?>
```

verrà visualizzato: `Ciao Anna!`

$_POST

$_POST è una variabile di tipo array che contiene le variabili passate dalla pagina precedente alla pagina attuale, tramite il method "post".

$_COOKIE

È un array che contiene le variabili passate, dalla pagina precedente a quella attuale, tramite cookie (un cookie è un file di testo salvato temporaneamente nel PC dell'utente che apre la pagina). Questa tecnica funziona solo se il browser utilizzato accetta cookie.

setcookie()-- crea un cookie (un file di testo temporaneo)

```
bool setcookie ( string $filename [, string $text [, int $expire
[, string $path [, string $domain [, bool $secure [, bool
$httponly ]]]]]] )
```

Funzione che inserisce, nell'header HTML della pagina ricevuta dall'utente, la richiesta di creare un file cookie nel PC dell'utente. Come le altre richieste del genere fatte negli header HTML al browser, deve essere fatta prima di qualsiasi funzione di output dello script PHP, compresi i tag <html> .

Dopo la chiamata della funzione `setcookie()` le altre pagine PHP possono accedere ai cookie così creati, usando l'array superglobale $_COOKIE.

variabile	contenuto
`$filename`	è l'unico parametro obbligatorio della funzione;
`$text`	indica il contenuto da salvare nel file; di solito è una stringa alfanumerica che identifica in modo univoco l'utente, oppure contiene delle variabili e dei valori che sono conservati per la durata di una sessione.
`$expire`	se impostato a zero significa alla chiusura del browser

esempio

```
setcookie ( "nomedelcookie", "contenuto", 0);
```

$_SERVER

È un array che contiene alcune variabili di sistema create dal server. Alcuni server web non forniscono tutte le informazioni previste.
Tra quelle più note vi sono:

$_SERVER['PHP_SELF'] contiene il percorso della pagina php attuale

$_SERVER['*HTTP_USER_AGENT*'] contiene il nome del browser (client)

$_SERVER["REMOTE_ADDR"] contiene l'indirizzo del client

$_SERVER['HTTP_CLIENT_IP'] ???

$_SESSION

È un array che contiene le variabili di sessione, cioè quelle create attraverso un cookie di sessione. Le variabili permettono di stabilire la continuità del collegamento tra l'utente e il server. Questa tecnica funziona solo se il browser utilizzato accetta cookie di sessione.

session_start() -- Inizializza i dati di una sessione

```
bool session_start ( void )
```

`session_start()` crea una sessione <u>oppure</u> riprende la sessione in corso; la sessione può essere continuata da una pagina all'altra tramite:

- il passaggio della costante di sistema `SID` (session id)
- i metodi "get" o "post"
- un cookie.

Per assegnare eventuali nomi alle sessioni, usare `session_name()` prima di `session_start()`.

SID (Session ID) è una costante di sistema che contiene l'identificatore di sessione passato tramite URL o tramite un cookie di sessione

~~Quando nel server sono~~ state ~~attivate le sessioni di transazione~~ session_start()~~registrerà il gestore dell'output interno per la riscrittura dell'URL.~~
Valore restituito
Questa funzione restituisce sempre TRUE.
Usare `session_start()` prima di qualsiasi altra funzione di output.

Esempio di avvio di sessione: page1.php

```
<?php

// page1.php
session_start();
$_SESSION['colore'] = 'verde';
$_SESSION['animale']   = 'gatto';
$_SESSION['istante']     = time();

// funziona solo se il browser accetta cookie di sessione
echo '<p><a href="page2.php">link page 2</a></p>';

// invia Session ID, se usato
echo '<p><a href="page2.php?'. SID .'">page 2</a></p>';

?>
```

Dopo aver visto page1.php, che avvia la sessione con `session_start()`, anche la seconda pagina, page2.php, potrà accedere ai dati della sessione.

Example #2 esempio di ripresa della sessione: page2.php

```
<?php

// page2.php
session_start();
echo $_SESSION['colore']; // verde
echo $_SESSION['animale'];   // gatto
echo date('Y m d H:i:s', $_SESSION['time']);

// in questo esempio si puo' usare anche SID,
// come fatto in page1.php

echo '<p><a href="page1.php">page 1</a></p>';

?>
```

Funzioni PHP per l'uso di un database di SQLite3

http://it2.php.net/manual/en/ref.pdo-sqlite.connection.php

A partire dalla versione 5.3 del linguaggio PHP è possibile utilizzare la notazione della programmazione orientata agli oggetti e l'interfaccia PDO (PHP Data Objects) per l'accesso ai file di SQLite3.

http://it2.php.net/manual/en/book.pdo.php

SQLite3 non è un DBMS server e si può usare solo in un file locale, ovvero, SQLite3 salva ogni database in un file, che deve essere sullo stesso computer dove si trovano le pagine php.

```
//crea un oggetto PDO e lo memorizza nella variabile $db
$db = new PDO("sqlite:/var/www/public/database.sq3");

//esegue un comando che non restituisce nulla
$db->exec("CREATE TABLE tabellina (cf CHAR(16) PRIMARY KEY);");

//esegue una interrogazione che restituisce un oggetto espressione
$espressione = $db->query("SELECT * FROM tabellina;");

//estrae il risultato dall'oggetto espressione
// nota: usare fetchAll() non usare fetch_all()
$risultato = $espressione->fetchAll();

echo "<pre>"; print_r($risultato);echo "</pre>";
```

Funzioni PHP per la comunicazione con un server DBMS

A partire dalla versione 5.3 del linguaggio PHP è possibile utilizzare la notazione della programmazione orientata agli oggetti e l'interfaccia PDO (PHP Data Objects) per la connessione al DBMS PostgreSQL e MySQL.

http://it2.php.net/manual/en/ref.pdo-pgsql.connection.php

http://it2.php.net/manual/en/ref.pdo-mysql.connection.php

```
<?php
  $db = new PDO("pgsql:
                            host=127.0.0.1;
                            port=5432;
                            dbname=esempio;
                            user=mario;
                            password=pass");
?>
```

```
<?php
  $db = new PDO("mysql:
                            host=127.0.0.1;
                            port=3307;
                            dbname=esempio;
                            user=mario;
                            password=pass" );
?>
```

TO DO: Character Encoding

PHP's XML extension supports the Unicode character set through different character encodings. There are two types of character encodings, source encoding and target encoding. PHP's internal representation of the document is always encoded with *UTF-8*.

The default source encoding used by PHP is *ISO-8859-1*
Currently, this means that such characters are replaced by a question mark.
http://php.net/manual/en/function.mb-internal-encoding.php

funzione per convertire dati sconosciuti in utf8

```
<?php
//$s is a string from whatever source
mb_detect_encoding($s, "UTF-8") == "UTF-8" ? : $s =
utf8_encode($s);
?>
```

http://it.php.net/manual/en/session.idpassing.php

```
string htmlentities(string $riga)
  permette di visualizzare correttamente $riga in una pagina HTML
```