La programmazione Web
4bis-Programmare in PHP

L'indirizzo della versione aggiornata dell'opera "Programmare in PHP" è: <http://www.urcanet.it/brdp/php_manual/>
Script di esempio sono contenuti all'indirizzo <*http://www.urcanet.it/brdp/php_manual/esempi/*>
L'opera è inclusa in "Appunti di informatica libera" di Daniele Giacomini.
La diffusione di questo documento è incoraggiata in base ai termini della licenza.

4b.1) Prefazione

Questo documento vuole essere una guida all'utilizzo di uno dei più versatili e potenti linguaggi di programmazione oggi diffuso in Internet.
Verranno risolti i problemi più comuni, progettati e realizzati i servizi necessari a un sito per essere moderno, dinamico e al passo coi tempi. Si pensi alla gestione di utenze, aree riservate, notizie, sondaggi, motori di ricerca, album fotografici, servizi FTP, ecc. Per fare questo si partirà da esempi molto semplici, scritti in modo altrettanto semplice, fino ad arrivare alla realizzazione di «classi» da utilizzare in tutti i nostri servizi. Prima di iniziare converrà collegarsi al sito <http://www.php.net> per scaricare il manuale ufficiale del PHP.
PHP sta per *Hypertext Preprocessor*. Dalla documentazione ufficiale il PHP viene definito come un «linguaggio script dal lato del servente immerso nel codice HTML La definizione nel dettaglio: *«linguaggio script»: il PHP è un vero e proprio linguaggio di programmazione, è importante rendersi conto che l'HTML, ad esempio, non è un linguaggio di programmazione ma un linguaggio di descrizione e formattazione del testo. Inoltre, per i più esperti, il PHP è un linguaggio interpretato.*
«Dal lato del servente»: queste le parole fondamentali. Il codice PHP inserito tra i marcatori HTML viene interpretato dall'elaboratore servente e non dal navigatore del cliente. Il modo migliore per rendere l'idea è passare ad un esempio.

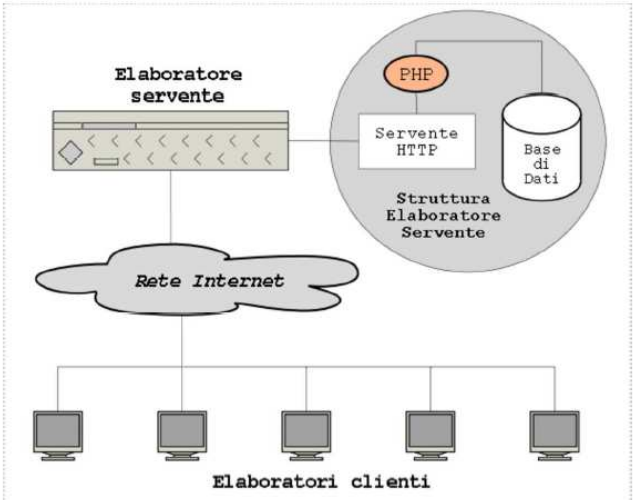
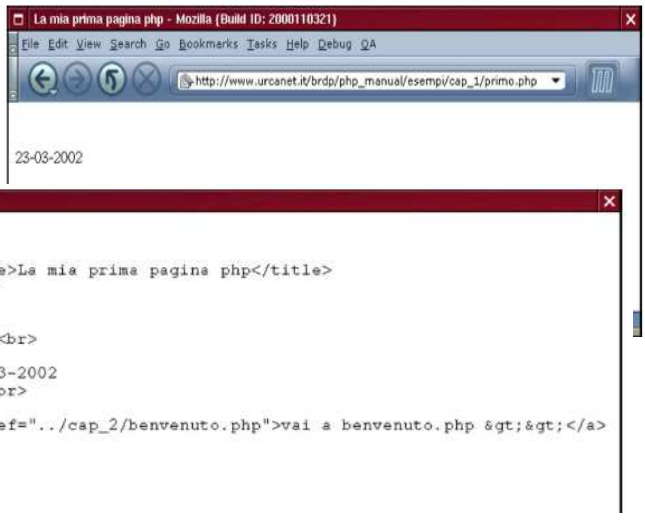


FIGURA: Il servente elabora tramite l'interprete PHP gli script i quali generano le pagine HTML che vengono fornite al cliente tramite il protocollo HTTP.

Si supponga di voler scrivere la data odierna in una pagina HTML. L'HTML non permette di farlo, in quanto, è un linguaggio di formattazione statico. Una soluzione potrebbe essere l'utilizzo di uno script Java di Netscape (JavaScript) ma è un linguaggio lato cliente. Il codice JavaScript viene eseguito dal navigatore del visitatore e non dal servente. Tutto ciò potrebbe portare alla visualizzazione di una data errata sulle pagine del sito solo perchè l'orario di sistema del visitatore non è impostato correttamente.
A questo punto l'ideale è fornire al navigatore una stringa HTML generata e formattata da una funzione PHP:

```
1.  <html>
2.  <head>
3.  <title>La mia prima pagina php</title>
4.  </head>
5.  <body>
6.  <br><br>
7.  <?php echo date("d-m-Y") ?>
8.  <br><br>
9.  <a href="benvenuto.php">vai a
    benvenuto.php &gt;&gt;</a>
10. </body>
11. </html>
```

Interessante è osservare il sorgente della



pagina dal menu *visualizza* del navigatore, ecco quello che viene inviato dal servente:

Una volta ricevuta la richiesta della pagina da parte del navigatore, il servente inizia a leggere il file '`primo.php`' e a inviarlo al cliente, appena trovati i marcatori '**`<?php ?>`**' capisce che deve farsi «tradurre» il codice in essi contenuto. Lo passa quindi all'interprete PHP che lo esegue e restituisce il risultato. In questo semplice esempio il risulato dell'operazione è la stringa '**`23-03-2002`**'.
La funzione utilizzata nell'esempio '**`date("d-m-Y")`**' formatta la data nello schema ***giornomese-anno*** (numerico), mentre '**`echo`**' visualizza a schermo il risultato della funzione '**`date()`**'.
Una modifica interessante al file precedente è: **`<?php phpinfo() ?>`**
Questa volta il risultato è una pagina che visualizza la configurazione dell'elaboratore servente, dell'interprete e dei componenti per l'interrogazione delle basi di dati, oltre a una serie di variabili predefinite. È bene dare un'occhiata al sorgente HTML della pagina, tutta opera dell'interprete PHP.

4b.2) Cosa può fare il PHP

Dunque il PHP permette di rendere dinamiche le pagine HTML dal lato del servente. Questo insieme alla sua semplicità di utilizzo, alla somiglianza con la sintassi C e alle innumerevoli funzioni che permettono di interagire con basi di dati e servizi di ogni genere ne hanno decretato il successo in Internet. La diffusione è dovuta alla semplicità di configurazione e alla completa compatibilità con numerosi serventi HTTP. Il PHP permette, in modo semplice e diretto, di far interagire il cliente con le basi di dati ospitate sul servente tramite il semplice utilizzo di un comune navigatore. Tutti i maggiori DBMS sono gestiti da PHP, senza dover installare software aggiuntivo o commerciale. Eccone un breve elenco:

Queste le basi di dati gestite dal PHP.
Mysql
Adabas D InterBase PostgreSQL
dBase FrontBase Solid
Empress mSQL Sybase
FilePro (read-only) Direct MS-SQL Velocis
IBM DB2 MySQL Unix dbm
Informix ODBC Ingres
Oracle (OCI7 and OCI8)

Inoltre il PHP permette di interagire con numerosi servizi tramite i protocolli: IMAP, SNMP, NNTP, POP3, HTTP e molti altri.
Il PHP è compatibile con molte piattaforme di sviluppo, tra cui quella Microsoft. Ovviamente le migliori prestazioni si ottengono in ambiente GNU/Linux (o più in generale su sistemi Unix).
Il PHP viene distribuito nei termini della: **The PHP License, version 2.02**.

4b.3) Le basi del linguaggio : la sintassi

In realtà '**`<?php ?>`**' non sono gli unici marcatori per fornire codice all'interprete. Ecco tutti i modi di includere codice PHP negli script:

- `<?` ***codice*** `?>`
- `<?php` ***codice*** `?>`
- `<script language="php">` ***codice*** `</script>`
- `<%` ***codice*** `%>`

Il primo è disponibile solo se sono stati impostati i marcatori abbreviati, ciò può essere fatto abilitando nel file di configurazione '`php.ini`' l'opzione '**`short_open_tag`**'. Per i particolari sugli altri marcatori si rimanda al manuale ufficiale.
Il primo concetto base è che le variabili in PHP iniziano con il carattere '**`$`**' e sono sensibili a maiuscole e minuscole, inoltre non è necessario definirne il tipo, sarà quello del primo dato ad essa assegnato (variant). Il tipo assegnato ad una variabile (qualora servisse) può essere ottenuto mediante la funzione `gettype($var).` E' possibile cambiare il tipo di dati mediante l'istruzione: `settype($variabile, tipo_variabile).`
Un secondo concetto è che per visualizzare il contenuto di una variabile si può utilizzare la sintassi: `<?=$nome ?>` o anche `<? echo $nome ?>`

Attenzione: se il codice PHP è formato da più istruzioni consecutive è richiesto il carattere punto e virgola (‘**;**’) alla fine di ognuna di esse, se invece l’istruzione da inserire è una sola, il ‘**;**’ può essere omesso.

Sarebbe buona norma dedicare molta cura all’indentazione del codice e inserire commenti chiari e diretti; così facendo si rende il codice più leggibile e comprensibile. La gestione dei commenti è simile a quella utilizzata in C, C++ e shell di Unix. I caratteri di commento sono: per una singola riga ‘**//**’ la barra obliqua doppia oppure il carattere ‘**#**’, per più righe il commento viene aperto con ‘**/***’ e chiuso con ‘***/**’. Nel secondo caso bisogna fare attenzione a non annidare i commenti.

```
0 <?
1 // Questo è un commento su una singola riga
2 // adesso visualizzo ciao mondo!
3
4 echo "ciao mondo!";
5
6 /*
7 questo è un commento
8 su più righe, posso scrivere di tutto!!!
9 anche istruzioni php che non verranno interpretate come:
10 echo "questo non verrà visualizzato!";
11 chiudo il commento
12 */
13
14 echo "<br>questo invece si vedrà!!";
15
16 /*
17 Notare che ho inserito <br> un marcatore html nel
18 codice php! Posso farlo.
19 */
20 ?>
```

4b.4) Il primo script

Come primo esempio verrà realizzata una pagina di riconoscimento del visitatore. Il PHP fornisce una serie di «variabili predefinite». Tra esse ce ne sono alcune fornite dal protocollo HTTP. Per l’elenco completo delle variabili predefinite è bene consultare il manuale ufficiale del PHP, molte di esse vengono visualizzate nella pagina generata dalla funzione ‘**phpinfo()**’.
Per questo semplice script verranno utilizzate:

- ‘**SERVER_NAME**’: nome dell’elaboratore su cui risiede e viene eseguito lo script;
- ‘**HTTP_REFERER**’: l’URI, se ne esiste uno, da cui proviene il visitatore;
- ‘**HTTP_USER_AGENT**’: il tipo di navigatore utilizzato dal visitatore;
- ‘**REMOTE_ADDR**’: l’indirizzo IP dell’elaboratore cliente;
- ‘**SCRIPT_NAME**’: il percorso dello script in esecuzione;

è importante fare molta attenzione all’utilizzo di queste variabili, alcune di esse usate in modo superficiale potrebbero fornire informazioni utili ai malintenzionati. Ad esempio ‘**DOCUMENT_ROOT**’ fornisce la directory su file system in cui si trovano gli script PHP e i documenti del servizio HTTP. Informazioni di questo tipo è meglio tenerle riservate!
Scelte le variabili va creato il nuovo file e salvato con il nome di ‘`benvenuto.php`’. La prima versione, la più rudimentale, è la seguente:

```
1 <html>
2 <head>
3 <title>Benvenuto!</title>
4 </head>
5 <body>
6
7 <br> La mia prima vera pagina in PHP.<br><hr>
8
9 <br><br> Informazioni sul server:
10 <br>Sei giunto su: <?=$SERVER_NAME ?>
12 <br>Stai eseguendo lo script: <?=$SCRIPT_NAME ?>
13
14 <br><br> Esaminiamo il client:
15
16 <br> Indirizzo IP: <?=$REMOTE_ADDR ?>
17 <br> Vedo che provieni da: <?=$HTTP_REFERER ?>
18 <br> Tipo di browser: <?=$HTTP_USER_AGENT ?>
19
20 </body>
21 </html>
```

Salvato il file nella directory del servizio HTTP e richiamato dal navigatore si ottiene qualcosa simile a:

```
La mia prima vera pagina in PHP.
Informazioni sul server:
Sei giunto su: www.urcanet.it
Stai eseguendo lo script: /brdp/php_manual/esempi/cap_2/benvenuto.php
Esaminiamo il client:
Indirizzo IP: 127.0.0.1
Vedo che provieni da: http://www.urcanet.it/brdp/php_manual/esempi/cap_1/primo.php
Tipo di browser: Mozilla/5.0 (X11; U; Linux 2.4.2-2 i686; en-US; 0.7) Gecko/20010316
```

4b.5) Tipi di dati

Il PHP gestisce quattro tipi ***scalari***, due tipi ***composti*** e due tipi ***speciali*** di dati.

Scalari	Composti	Speciali
'**boolean**'	'**array**'	'**resource**'
'**integer**'	'**object**'	'**NULL**'
'**double**'		
'**string**'		

Come già accennato in precedenza, il tipo di una variabile può essere modificato in ogni momento con la funzione '**settype($variabile,*tipo*)**' che restituisce un valore booleano vero o falso rappresentante l'esito dell'operazione. I possibili tipi da assegnare alla variabile sono:

- '**integer**'
- '**double**'
- '**string**'
- '**array**'
- '**object**'

I criteri di conversione dei tipi sono molto importanti, il codice mostrato di seguito non provoca problemi.

```
0 <?
1 $a = 231; // da questo momento in poi $a è un intero
2 if(settype($a,double)){
3       /*
4       Se l'operazione è riuscita il valore è TRUE quindi
5       visualizza il testo seguente. Altrimenti salta alla
6       fine dell'if riga 10.
7       */
8       echo "<br>valore settato a double. Ecco il nuovo valore: ";
9       echo $a;
10 }
11 ?>
```

Spiacevoli inconvenienti potrebbero verificarsi se si prova a convertire una stringa in un intero. Un'altra funzione utile per la gestione dei tipi di dati è '**gettype(*$variabile*)**' che restituisce una stringa contenente il tipo della variabile a essa fornita. Questi i possibili valori restituiti dalla funzione:

- '**integer**'
- '**double**'
- '**string**'
- '**array**'
- '**object**'
- '**unknown type**'

4b.5.1) Integer

Il tipo di dati '**integer**' rappresenta tutti i numeri dell'insieme matematico degli interi.

$$Z = \{..., -2, -1, 0, 1, 2, ...\}$$

Quindi tutti i numeri interi, sia negativi che positivi, vengono gestiti dal PHP come tipi '**integer**'.
Anche i corrispondenti valori in base otto e sedici vengono gestiti come gli interi. Ecco alcuni esempi tratti dal manuale ufficiale:

```
<?
$a = 1234; // numero intero positivo
$a = -123; // numero intero negativo
$a = 0123; // numero ottale (equivalente a 83 in decimale)
$a = 0x1A; // numero esadecimale (equivalente a 26 in decimale)
?>
```

Il limite massimo della dimensione dei valori interi dipende dall'architettura dell'elaboratore su cui si lavora.
Nel PHP non esiste un tipo specifico per i numeri naturali. Essi vengono gestiti come '**integer**'.

Esiste la funzione, '**is_integer()**', che restituisce un valore booleano vero se la variabile ad essa fornita è di tipo intero, falso se la variabile non è intera.

4b.5.2) Boolean

Il tipo di dati '**boolean**' può assumere solo due valori, vero o falso.
Questo tipo viene utilizzato frequentemente per la gestione degli errori. Molte funzioni, infatti, restituiscono un valore booleano che assume valore vero se l'operazione è andata a buon fine, falso se si è verificato un errore.
Per assegnare il valore vero o falso a una variabile è sufficiente l'istruzione:

```
$bit = True;
```

In questo esempio alla variabile '**$bit**' è stato assegnato il valore vero. Anche se l'istruzione '**if**' non è ancora stata trattata, è necessario anticipare le metodologie di controllo sul valore della variabile.
In PHP non è necessario (anche se è possibile) eseguire il controllo come illustrato nel primo esempio.

L'interprete riconosce il tipo di dato e controlla se è vero. E' quindi ammesso anche lo script a fianco.

```
<?
$bit = True;
if($bit == "True"){
// $bit è vero
echo   "   il   valore   è
vero!";
}
?>
```

```
<?
$bit = True;
if($bit){
// $bit è vero
echo   "   il   valore   è
vero!";
}
?>
```

4b.5.3) Double

Il PHP non necessita della definizione dei tipi di dati che si vanno a utilizzare ma inizializza il tipo di variabile in base al primo valore a essa associato. I «numeri a virgola mobile» in PHP contengono i tipi conosciuti in altri linguaggi come: '**floats**', '**doubles**', '**real**'. È possibile inizializzare una variabile di tipo '**double**' assegnandole valori formattati in uno dei seguenti modi:

```
<?
$a = 1.234;
$a = 1.2e3;
$a = 7E-10;
?>
```

Anche per questo tipo, come per gli interi, la dimensione massima dipende dall'architettura dell'elaboratore su cui viene eseguito lo script. La funzione '**is_double()**' serve per testare se una variabile è di tipo double.

4b.5.4) String

Una stringa è un insieme di caratteri. In PHP non ci sono limiti particolari di lunghezza per i dati di tipo '**string**'. Ci sono più modi di delimitare una stringa. Ecco i due più comuni:

```
0 <?
1 // single quoted string
2 $nome = 'Gianluca Giusti';
3
4 // double queoted string
5 $nome = "Gianluca Giusti";
6
7 /* In entrambi i casi $nome viene inizializzata
8 come stringa dall'interprete PHP */
9 ?>
```

L'interprete PHP riconosce come carattere di escape il '\' (barra obliqua inversa). Di seguito riportiamo i caratteri più comuni, una tabella completa è contenuta nel manuale ufficiale.

Alcuni caratteri di escape per le stringhe PHP.

Carattere e	Significato
\t	carattere tabulazione
\n	carattere di fine riga
\\	'****' barra obliqua inversa
\"	'**"**' doppio apice
\$	'**$**' dollaro

Ecco alcuni esempi di come gestire le stringhe.

```
0 <?
1 // Valorizzo una variabile che servirà in seguito.
2 $email = " brdp @ urcanet.it ";
3
4 // Stringa semplice:
5 $stringa = "Ecco. Questa è una stringa";
6
7 // Ad essa si possono concatenare altre stringhe alla
fine
8 $stringa = $stringa." con dell'altro testo aggiunto";
9
10 // che equivale a scrivere
11 $stringa .= " con dell'altro testo aggiunto";
```

```
12
13 // Oppure all'inizio
14 $stringa = "altro testo ancora ".$stringa;
15
16 // Adesso si prende la prima lettera della stringa
17 $prima = $stringa{0};
18
19 // Concatenazione multipla
20 $stringa = $prima." - ".$stringa." <br><br> il mio email: ".$email;
21
22 /* Test sui caratteri di escape. All'interno della stringa posso
23 visualizzare il valore di una variabile ma per visualizzarne
24 il nome devo "inibire" il carattere speciale $. Lo faccio con
25 l'uso della barra obliqua inversa. Ecco come: */
26 $stringa = "Questo il valore di \$email: $email";
27
28 // c'è differenza con:
29 $stringa = "Questo il valore di $email: $email";
30 ?>
```

Il PHP fornisce un gran numero di funzioni dedicate alla gestione delle stringhe; uno dei punti di forza di questo linguaggio sta proprio nella semplicità con cui queste possono essere manipolate. (Vedi Sezione «String functions» del manuale). La funzione '**`is_string()`**' serve per testare se una variabile è di tipo string.

4b.5.5) Array

Gli array possono essere creati tramite il costrutto '**`array()`**' oppure tramite la valorizzazione degli elementi.
A differenza dei classici linguaggi di programmazione, il PHP permette di indicizzare gli array non solo mediante interi non negativi, ma anche tramite stringhe. Per chi conosce il Perl il concetto è molto simile a quello degli array associativi. La funzione '**`is_array()`** serve per testare se un avriabile è di tipo array.
Di seguito la sintassi per creare un array tramite il costrutto '**`array()`**'.

```
0 <?
1 array( [chiave =>] valore
2 , ...
3 )
4 // la chiave può essere un intero non negativo o una stringa
5 // il valore può essere qualunque
6 ?>
```

La chiave è contenuta tra le parentesi quadre perchè può essere omessa, se non viene specificata viene incrementato il valore intero. La sintassi di associazione è molto semplice ed intuitiva, per le chiavi intere positive: '***chiave* `=>` *valore***', mentre per le chiavi di tipo stringa vanno aggiunte le doppie virgolette '**`"`*chiave*`" =>` *valore***'.
Per visualizzare il contenuto dell'array è possibile utilizzare la semplice istruzione '**`print_r($array)`**'. Negli esempi seguenti si vedrà come questa funzione visualizza l'array.

```
0 <?
1 $a = array( "a", "b", 44, "d", "e");
2 print_r($a);
3 ?>
```

L'istruzione '**`print_r($a)`**' visualizzerà sullo schermo la struttura e il contenuto dell'array nel seguente modo:

```
Array ( [0] => a [1] => b [2] => 44 [3] => d [4] => e )
```

L'indice dei valori è stato incrementato automaticamente. È bene tenere sempre presente che le chiavi degli array partono da 0 e non da 1.
C'è la possibilità di specificare solo alcune chiavi e lasciare la gestione degli indici omessi all'interprete.

```
0 <?
1 $a = array( "a", "b", "c", "d", 8=>"e", 4=>"f", "g", 3=>"h");
2 print_r($a);
3 ?>
```

Questo il risultato dello script precedente:

```
Array ( [0] => a [1] => b [2] => c [3] => h [8] => e [4] => f [9] => g )
```

A prima vista il risultato può sembrare errato, ma non è così. L'interprete incrementa automaticamente gli indici omessi dei primi quattro valori, ossia da 0 a 3, e mano a mano valorizza l'array. La lettera '**e**' va invece inserita nella cella con chiave, specificata, pari a 8 e la '**f**' in quella con chiave 4. Per la lettera '**g**' non viene specificata nessuna chiave, l'interprete, quindi, riparte automaticamente dall'indice intero più alto incrementandolo di uno e ottiene il valore 9. Rimane da inserire l'ultima lettera '**h**' che va, come richiesto, nella cella con chiave pari a 3. Questa operazione sovrascrive la lettera '**d**' che già era stata inserita, in automatico, con chiave pari a 3. A questo

punto dovrebbero essere chiari i motivi dell'assenza della lettera '**d**' e del particolare ordine con cui vengono visualizzati i valori contenuti nell'array.
Un array può essere creato anche senza l'utilizzo del costrutto '**`array()`**', in questo caso la sintassi ricorda quella dei più comuni linguaggi di programmazione.

```
0 <?
1 $a[] = "a";
2 $a[] = "b";
3 $a[] = 44;
4 $a[] = "d";
5 $a[] = "e";
6 print_r($a);
7 ?>
```

Il secondo esempio può essere tradotto in questo modo:

```
0 <?
1 $a[] = "a";
2 $a[] = "b";
3 $a[] = "c";
4 $a[] = "d";
5 $a[8] = "e";
6 $a[4] = "f";
7 $a[] = "g";
8 $a[3] = "h";
9 print_r($a);
10 ?>
```

I due procedimenti per la creazione degli array sono equivalenti.
Gli array possono avere chiavi miste, come nell'esempio seguente, in cui alcune sono intere e alcune stringhe.

```
0 <?
1 $a["rosso"] = "a";
2 $a[] = "c";
3 $a[8] = "e";
4 $a["nero"] = "f";
5 $a[] = "g";
6 print_r($a);
7 ?>
```

Ecco il risultato:

```
Array ( [rosso] => a [0] => c [8] => e [nero] => f [9] => g )
```

È interessante studiare una possibile applicazione pratica degli array. Nel seguente esempio verrà ripresa la funzione '**`date()`**' già incontrata precedentemente.

```
0 <?
1 // valorizzo l'array dei giorni della settimana con il metodo classico.
2 $giorno[0] = "Domenica";
3 $giorno[1] = "Lunedì";
4 $giorno[2] = "Martedì";
5 $giorno[3] = "Mercoledì";
6 $giorno[4] = "Giovedì";
7 $giorno[5] = "Venerdì";
8 $giorno[6] = "Sabato";
9
10 // valorizzo l'array dei mesi dell'anno con il costrutto array()
11 $mese = array(
12 1 => "Gennaio",
13 2 => "Febbraio",
14 3 => "Marzo",
15 4 => "Aprile",
16 5 => "Maggio",
17 6 => "Giugno",
18 7 => "Luglio",
19 8 => "Agosto",
20 9 => "Settembre",
21 10 => "Ottobre",
22 11 => "Novembre",
23 12 => "Dicembre"
24 );
25 // Prendo il mese in formato numerico da 1 a 12.
26 $numero_mese = date("n");
27
28 /* Prendo il giorno della settimana da 0 (domenica) a 6 (sabato)
29 questa volta formatto tutto annidando più funzioni.
30 in PHP è possibile! */
31 $giorno_settimana = $giorno[date("w")];
32
33 // Formatto la data nel modo: Lunedì 19 Novembre 2001
34 $oggi = $giorno_settimana." ".date("d")."-".$mese[$numero_mese]."-".date("Y");
35
36 // Visualizzo la data a schermo concatendandola ad una stringa
```

```
37 echo "<br> Oggi è: <b>".$oggi."</b>";
38 ?>
```

Il risultato di questo scriptè:

```
Oggi è: Domenica 18-Novembre-2001
```

Gli array rappresentano una delle strutture più versatili in PHP, a differenza dei linguaggi classici, infatti, la dimensione non deve essere specificata a priori. Questo permette una dinamicità e una libertà di utilizzo notevole.

Il PHP gestisce anche gli array multidimensionali e gli array annidati.

Non si entrerà nel merito, si darà per scontata la teoria sulla gestione degli array, che è simile per i diversi tipi di linguaggi, mentre si tratterà la sintassi tramite uno script di esempio.

Il seguente è un esempio sull'utilizzo degli array multidimensionali.

```
0 <html>
1 <head>
2 <title>Semplice Agenda telefonica statica</title>
3 </head>
4 <body>
5
6 <?
7 /*
8 Un semplice esempio di array multidimensionale.
9 Una rubrica telefonica.
10 */
11
12 $a["nome"][0] = "Gianluca";
13 $a["cognome"][0] = "Giusti";
14 $a["tel"][0] = "06/66666666";
15
16 $a["nome"][1] = "Mirko";
17 $a["cognome"][1] = "Simeoni";
18 $a["tel"][1] = "07/77777777";
19
20 $a["nome"][2] = "Fabio";
21 $a["cognome"][2] = "Ferri";
22 $a["tel"][2] = "08/88888888";
23
24 /*
25 Adesso elenchiamo la rubrica. Lo faremo senza
26 utilizzare Nessuna struttura ciclica per non
27 confondere le idee
28 */
29
30 ?>
31
32 <br>
33
34 <table border="1">
35 <tr bgcolor="gray" >
36 <td>ID</td>
37 <td>Nome</td>
38 <td>Cognome</td>
39 <td>Telefono</td>
40 </tr>
41 <tr>
42 <td>0</td>
43 <td><?=$a[nome][0]?></td>
44 <td><?=$a[cognome][0]?></td>
45 <td><?=$a[tel][0]?></td>
46 </tr>
47 <tr>
48 <td>1</td>
49 <td><?=$a[nome][1]?></td>
50 <td><?=$a[cognome][1]?></td>
51 <td><?=$a[tel][1]?></td>
52 </tr>
53 <tr>
54 <td>2</td>
55 <td><?=$a[nome][2]?></td>
56 <td><?=$a[cognome][2]?></td>
57 <td><?=$a[tel][2]?></td>
58 </tr>
59
60 </table>
61
62 </body>
```

```
63</html>
```

4b.5.6) Object

Una classe è una collezione di variabili e di funzioni dette metodi. Una volta definita la classe è possibile istanziare uno o più oggetti della stessa classe. Ognuno di essi è indipendente dagli altri.
Una trattazione approfondita può essere trovata più avanti nella trattazione.

4b.6) Operatori

Il PHP gestisce numerosi tipi di operatori: aritmetici, assegnazione, controllo degli errori logici, incremento e decremento confronto.

4b.6.1) Operatori aritmetici

Le caratteristiche degli operatori aritmetici.

$a + $b	Addizione Somma tra $a e $b
$a - $b	Sottrazione Differenza tra $a e $b
$a * $b	Moltiplicazione Prodotto tra $a e $b
$a / $b	Divisione Quoziente tra $a e $b

4b.6.2) Operatori di assegnazione

L'operatore fondamentale per l'assegnazione di un valore ad una variabile è '**=**'. La sintassi fondamentale è:

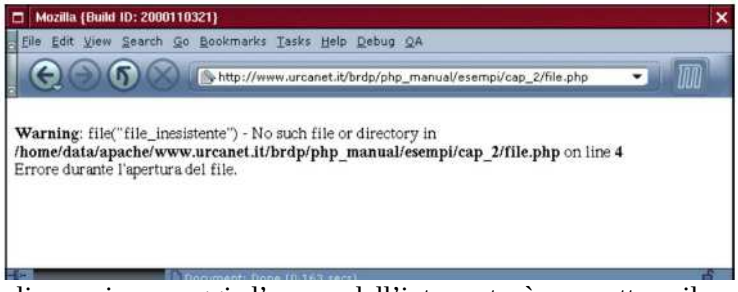
```
$a = 123;
$b = $a;
$c = "questa è una stringa";
$a = $a + $b;
$a += $b;
$a -= $b;
$a *= $b;
$a /= $b;
$a = $b = 30;
// o ancora in questo esempio sia $a che $b valgono 6
$a = 3; $b = $a += 3;
26 $a = "ciao a";
27 $a .= " tutti!!!";
```

4b.6.3) Operatori di controllo degli errori

Nell'esempio seguente viene gestito un errore dovuto al tentativo di lettura di un file inesistente. Con l'occasione viene utilizzata la funzione PHP '**file('nome_del_file')**' che restituisce un array contenente le righe del file passato alla funzione.

```
// provo ad aprire un file che non esiste
$file_array = file('file_inesistente') or die (" Errore durante l'apertura del file. ");
/* se il file non esiste, lo script si ferma mostrando i messaggi di errore */
```

Se si salva il codice in un file e si richiama tramite il navigatore, si nota che oltre al messaggio di errore inserito nello script viene visualizzato un messaggio introdotto dall'interprete PHP. I messaggi di errore possono contenere informazioni utili ai male intenzionati, è bene, quindi, gestire tali messaggi al meglio. Un esempio è riportato in figura dove si nota il percorso su file system dello script di prova. Un modo molto semplice per non far visualizzare i messaggi d'errore dell'interprete è premettere il carattere '@' alla funzione. Ad esempio: '**@file('dati.txt')**'. L'esempio precedente potrebbe essere modificato in questo modo:




```
$file_array = @file('file_inesistente') or die (" Errore durante l'apertura del file. ");
/* In questo caso il messaggio di errore dell'interprete PHP è stato silenziato dalla '@'
premessa alla funzione file(). Verrà quindi mostrato solo il nostro messaggio di errore */
```

Il costrutto '**die()**' non fa altro che terminare l'esecuzione del programma, in caso di errore, mostrando il messaggio contenuto tra le parentesi.

4b.6.4) Operatori di incremento e decremento

Come detto, la sintassi PHP ricorda molto quella del C. Gli operatori di pre e post incrementazione sono gestiti come nel linguaggio C.

'**$a++**'	Post-incremento Restituisce '**$a**' e poi la incrementa di uno
'**++$a**'	Pre-incremento Incrementa di uno '**$a**' e poi la restituisce
'**$a--**'	Post-decremento Restituisce '**$a**' e poi la decrementa di uno
'**--$a**'	Pre-decremento Decrementa di uno '**$a**' e poi la restituisce

4b.6.5) Operatori logici

Gli operatori logici gestiti dal PHP sono riportati in tabella .
Le precedenze degli operatori sono riportate nel manuale ufficiale.

'**$a and $b**'	AND vera se $a e $b sono vere
'**$a or $b**'	OR vera se $a o $b è vera
'**$a Xor $b**'	XOR vera se $a o $b è vera ma non entrambe
'**!$a**'	NOT Negazione. Vera se $a non è vera
'**$a && $b**'	AND Simile a '**and**' ma con precedenza diversa
'**$a || $b**'	OR Simile a '**or**' ma con precedenza diversa

4b.6.6) Operatori di confronto

'**$a == $b**'	Uguale vera se $a è uguale a $b
'**$a === $b**'	Identico vera se $a è uguale a $b e sono dello stesso tipo
'**$a != $b**'	Diverso vera se $a è diverso da $b
'**$a <> $b**'	Diverso vera se $a è diverso da $b
'**$a !== $b**'	Non Identico vera se $a non è uguale a $b o non sono dello stesso tipo
'**$a < $b**'	Minore vera se $a è minore di $b
'**$a > $b**'	Maggiore vera se $a è maggiore di $b
'**$a <= $b**'	Minore o uguale vera se $a è minore o uguale a $b
'**$a >= $b**'	Maggiore o uguale vera se $a è maggiore o uguale a $b

4b.7) Strutture di controllo

Strutture di controllo utilizzabili:

if while break else do..while include() elseif for switch foreach

4b.7.1) Struttura if

Il costrutto '**if**' è molto importante e utilizzato in tutti i linguaggi di programmazione. La sintassi è la seguente:

```
if(condizione)
singola istruzione
```

Utilizzato in questo modo, solamente un'istruzione è condizionata dall'esito della condizione contenuta nel costrutto '**if**'.

```
if($a < $b) echo "Condizionato da if. Solo se \$a è minore di \$b.";
```

Se si ha bisogno di condizionare una o più istruzioni, il blocco delle istruzioni condizionate va contenuto tra le parentesi '**{ }**'.

```
if(condizione){
        istruzione 1
        istruzione 2
        .
        istruzione n
}
```

Segue la sintassi del costrutto '**else**'.

```
if(condizione){
        istruzione 1
        istruzione 2
        .
        .
        .
        istruzione n
}else{
        istruzione 1
        istruzione 2
        .
        .
        .
        istruzione m
}
```

Si posso innestare più costrutti if mediante il costrutto elseif.

```
if(condizione){
        istruzione 1
        istruzione 2
        .
        .
        .
        istruzione n
}elseif(condizione){
        istruzione 1
        istruzione 2
        .
        .
        .
        istruzione m
}else{
        istruzione 1
        istruzione 2
        .
        .
        .
        istruzione k
```

```
}
```

4b.7.2) Struttura switch

Spesso ci si trova a dover confrontare il contenuto di una variabile con più valori, in questi casi è preferibile utilizzare la struttura '**switch**' al posto di '**elseif**'.

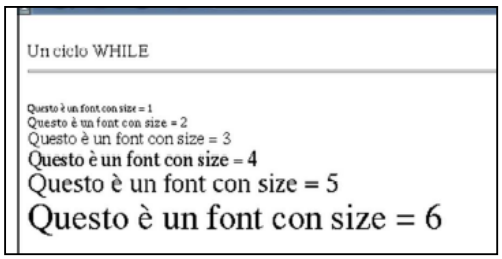
```
switch ($scelta) {
        case 0:
                <istruzioni>
                break; // esce da switch
        case 1:
                <istruzioni>
                break; // esce da switch
        case 2:
                <istruzioni>
                break; // esce da switch
        default:
                <istruzioni>
}
```

Nella struttura '**switch**' l'interprete esegue le istruzioni successive al '**case**' soddisfatto. Questo è il motivo per cui si utilizza il comando '**break**'.

4b.7.3) Struttura while

Il '**while**' è la struttura di gestione dei cicli più semplice del PHP

```
while(condizione){
istruzione 1
istruzione 2
.
istruzione n
}
```




```
<html>
<head>
<title>Ciclo while</title>
</head>
<body>
<br> Un ciclo WHILE<br><hr>
<?
$i=1;
while ($i <= 6){ ?>
 <br> <font size="<?=$i?>">
 Questo è un font con size = <?=$i?>
 </font>
<?
$i++;
}
?>
</body>
</html>
```

A differenza del '**while**' il '**do..while**' esegue il controllo sulla condizione dopo l'esecuzione delle istruzioni.

```
do {
        istruzione 1
        istruzione 2
        .
        istruzione n
}while(condizione);
```

Dunque l'interprete esegue le istruzioni e solo dopo controlla la condizione di stop, se è vera riesegue le istruzioni, se è falsa abbandona il ciclo.

4b.7.4) Struttura for

La sintassi del ciclo '**for**' :

```
for(condizione iniziale; condizione stop; variazione parametro){
        istruzione 1
        istruzione 2
        .
        istruzione n
}
```

Tale sintassi viene applicata nel seguente modo:

```
for($i=0; $i<10; $i++){echo "<br>tutte le istruzioni che desidero";}
```

Sia le condizioni che la variazione possono essere omesse o incluse tra le istruzioni. Per fare questo è necessario utilizzare la struttura '**break**'

4b.7.5) Struttura foreach

Va subito notato che il costrutto '**foreach**' è stato implementato dalla versione 4.0 in poi e quindi non è gestita dalle versioni precedenti del PHP. L'uso del '**foreach**' è indicato per la gestione degli array e mette a disposizione due sintassi principali:

```
foreach($variabile_array as $value){
        istruzioni
        ....
}
```

e:

```
foreach($variabile_array as $key => $value){
        istruzioni
        ...
}
```

il suo funzionamento, a differenza del semplice '**for**', non è molto intuitivo ed è vincolato all'utilizzo di un array. Si nota immediatamente che la sintassi non richiede l'utilizzo di indici né di incrementi, infatti il costrutto opererà su tutti gli argomenti dell'array indistintamente. Dunque questo strumento risulta comodo e versatile quando si ha a che fare con array con chiavi particolari o non consecutive (nel caso di chiavi numeriche).
Si supponga di avere un primo array composto da chiavi intere e consecutive simile al seguente:

```
<?
$array_1 = array (1, "a", 5, "c", "ciao"); // con chiavi da 0 a 4
print_r($array_1);
?>
```

e un secondo composto da chiavi miste e quindi non consecutive come questo:

```
<?
$array_2 = array (
"uno" => 1, // chiave stringa e valore intero
"frutta" => "mela", // chiave stringa e valore stringa
5 => "cinque", // chiave intero e valore stringa
40 => 30 // chiave intero e valore intero
);
print_r($array_2);
?>
```

I risultati dei due script sono semplicemente:

```
Array ( [0] => 1 [1] => a [2] => 5 [3] => c [4] => ciao )
Array ( [uno] => 1 [frutta] => mela [5] => cinque [40] => 30 )
```

Si supponga ora di non conoscere le chiavi del secondo array perchè assegnate dinamicamente da un codice precedente. In questo caso non si sarebbe in grado di operare sull'array con le strutture cicliche classiche perchè non c'è nessuna relazione tra le chiavi. Mentre nel primo la relazione tra le chiavi è semplice, sono interi successivi a partire da 0. In queste situazioni la struttura '**foreach**' è tanto indispensabile quanto semplice ed immediata da utilizzare.
L'esempio potrebbe continuare con la necessità di separare i valori interi da quelli non interi, contenuti in '**$array_2**'. Di seguito viene risolto il problema con l'utilizzo del costrutto '**foreach**' e della prima sintassi.

```
<?
foreach ($array_2 as $val){
        echo "<br>\$val = ".$val;
        if(is_integer($val)){
                echo " ------&gt; ed è un intero...";
        }else{
                echo " ------&gt; ma non è un intero...";
        }
}
?>
```

il risultato è il seguente:

```
$val = 1 ------> ed è un intero...
$val = mela ------> ma non è un intero...
$val = cinque ------> ma non è un intero...
$val = 30 ------> ed è un intero...
```

A questo punto è semplice dedurre il funzionamento della seconda sintassi, infatti, il '**foreach**' può estrarre sia la chiave che il valore degli elementi contenuti nell'array. Complicando ulteriormente lo script precedente si può verificare il funzionamento della seconda sintassi.

```
<?
foreach ($array_2 as $chiave => $valore) {
        echo "<br>\$val = ".$valore;
        if(is_integer($valore)){
```

```
                echo " ------&gt; ed è un intero...";
                echo " ------&gt; questa la sua chiave: ".$chiave;
        }else{
                echo " ------&gt; ma non è un intero...";
                echo " ------&gt; questa la sua chiave: ".$chiave;
        }
}
?>
```

Il risultato è il seguente:

```
$val = 1 ------> ed è un intero... ------> questa la sua chiave: uno
$val = mela ------> ma non è un intero... ------> questa la sua chiave: frutta
$val = cinque ------> ma non è un intero... ------> questa la sua chiave: 5
$val = 30 ------> ed è un intero... ------> questa la sua chiave: 40
```

4b.7.6) Costrutto break

Il costrutto '**`break`**' serve ad abbandonare una struttura di controllo, nel dettaglio permette di uscire da: '**`for`**', '**`foreach`**', '**`while`**', '**`do..while`**' e '**`switch`**', e ammette un parametro numerico opzionale. Tale parametro deve essere un intero che indicherà il "livello" della struttura da abbandonare. Ad esempio, nel caso di due o più strutture annidate, si potrebbe voler uscire da una o più di esse: questo è possibile tramite il parametro.
Con l'occasione verranno approfondite, negli esempi seguenti, le strutture tralasciate in precedenza.

```
<?
$i = 0;
while (++$i){ // il while incrementa anche in questo modo
        if($i == 3){
                echo "<br>Conto fino a ".$i." ed esco solo dal conteggio: ";
                for ($t = 1; ;$t++) { // la condizione di stop è tra le istruzioni.
                        if ($t > $i) {
                                break 1; // esce solo dal for equivale a break senza parametri.
                        }
                        echo " ".$t;
                }
        } elseif($i == 9){
                echo "<br> Conto fino a ".$i." ed esco dal for e dal while:";
                $t = 1;
                for (;;) { // Nessuna condizione è espressa nel costrutto.
                        if ($t > $i) {
                                break 2; // il parametro è 2 quindi esce dal for e dal while.
                        }
                        echo " ".$t;
                        $t++;
                }
        }
}
echo "<br><br> fine dello script!";
?>
```

4b.7.7) Costrutto include( )

Il PHP permette di includere uno o più file in un punto ben preciso di un altro file. Tali file possono, a loro volta, contenere codice PHP che verrà interpretato se delimitato dagli appositi marcatori.
Si pensi alla gestione di modelli grafici, alla necessità di dichiarare le stesse variabili in più file, all'inclusione di funzioni già pronte e via dicendo. In tutti questi casi è possibile creare un file specifico in cui inserire il codice che va incluso negli script. Ovviamente è possibile includere anche file residenti su macchine diverse dalla nostra. Per l'inclusione dei file il PHP offre diverse soluzioni. In questa guida verrà approfondita la funzione '**`include()`**'. Ne esistono altre che sono riportate nel manuale ufficiale.

```
<?
// inclusione di un file chiamato esterno.html
        include("esterno.html");
        include("http://www.url_altro_sito.com/esterno.html");
?>
```

Prelevare file da altre macchine è del tutto corretto. Tuttavia va tenuto conto dei seguenti fattori:

- l'elevato tempo di accesso al servente remoto, che può causare un ritardo nella generazione della pagina e può rallentare notevolmente la visualizzazione della stessa;
- l'elaboratore servente potrebbe essere irraggiungibile o il file essere inesistente;
- il contenuto del file remoto potrebbe essere modificato a nostra insaputa.

Il secondo punto può essere affrontato in vari modi, quello più sbrigativo è sicuramente la gestione dell'errore generato con la mancata inclusione del file. Ecco un modo di procedere:

```
<?
/* inclusione di un file remoto. Su una macchina diversa,tramite il protocollo HTTP.
 compresa la gestione dell'errore!!!
*/
if( !@include("http://www.url_altro_sito.com/esterno.html") )
        echo "<br><br> Problemi di visualizzazione! torna a trovarci...";
?>
```

Se il file remoto è uno script, ad esempio PHP, prima di essere incluso viene interpretato dal servente, quindi viene incluso il risultato dello script generato in remoto e non sull'elaboratore su cui è ospitato il nostro script .

4b.8) Passaggio di variabili tramite HTTP

Fino ad ora sono stati trattati i casi di singole pagine PHP in cui le variabili sono valorizzate, manipolate e visualizzate con l'interpretazione del singolo file. In questo modo l'utente che naviga la pagina non è in grado di interagire con gli script, non può passare dei dati al programma perché non può editarne il codice. Il protocollo HTTP permette, tramite i marcatori HTML di passare dei dati tra le pagine, il PHP è in grado di ricevere ed elaborare queste informazioni.

L'HTML permette al navigatore di inserire dati tramite il marcatore '**`form`**' che riceve dagli attributi '**`action`**' e '**`method`**' le direttive relative al file che deve ricevere i dati e al modo in cui essi devono esseregli passati. L'attributo '**`method`**' ammette due possibili valori: '**`get`**' e '**`post`**'.

4b.8.1) Passaggio di variabili : metodo GET

Quando si sceglie il metodo '**`get`**' le variabili ed il relativo valore vengono fornite allo script destinatario tramite la barra dell'indirizzo del *browser*.

Il modo migliore per chiarire il concetto è l'utilizzo di un esempio. Verranno realizzati due script, '`get_1.html`' e '`get_2.php`'. '`get_1.html`' invierà i dati contenuti nel '**`form`**' a '`get_2.php`' utilizzando il metodo '**`get`**', '`get_2.php`' riceverà i dati, li manipolerà e li visualizzerà in una semplice pagina di saluto.

Ecco il codice del file '`get_1.html`':

```
<html>
<head>
<title> Passaggio del nome! </title>
</head>
<body>
<br>
 Una semplice pagina HTML che passa nome e cognome ad
 uno script PHP che saluterà il navigatore.
 <br><br>
<form method="get" action="get_2.php">
 Dimmi il tuo nome: <input type="Text" name="nome"> <br><br>
 Ed ora il Cognome: <input type="Text" name="cognome"> <br><br>
 <input type="Submit" value="Adesso invia i dati in GET &gt;&gt;">
 </form>
</body>
</html>
```

Il primo file invia i due campi di testo al file destinatario '`get_2.php`' che riceve due variabili valorizzate con i dati inseriti dal navigatore. Le variabili hanno il nome del campo del '**`form`**', nell'esempio le variabili sono '**`$nome`**' e '**`$cognome`**' e sono di tipo stringa. Il codice del secondo script è molto semplice, non fa altro che visualizzare i valori delle due variabili.

```
<? $nome=$_REQUEST['nome']; $cognome=$_REQUEST['cognome']; ?>
<html>
<head>
<title> Pagina di destinazione... </title>
</head>
<body>
 Ciao, <br><br>
 Il tuo nome è: <?=$nome?> <br>
 e il tuo cognome è: <?=$cognome?>
</body>
</html>
```

L'obiettivo di questo esempio è quello di capire come funziona il metodo '**`get`**', per farlo basta leggere il contenuto della barra degli indirizzi nel *browser* durante la visualizzazione dello script destinatario. L'indirizzo è simile a:

`http://www.urcanet.it/ ... ual/esempi/cap_3/get_2.php?nome=Gianluca&cognome=Giusti`

Risulta semplice e intuitivo capire come vengono passate variabili e valori. All'indirizzo della

pagina viene aggiunto un carattere '**?**' e di seguito tutte le coppie di variabili e rispettivo valore separate dal '**=**', ad esempio: '**nome=Gianluca**'. Le coppie sono poi separate dal carattere '**&**'.

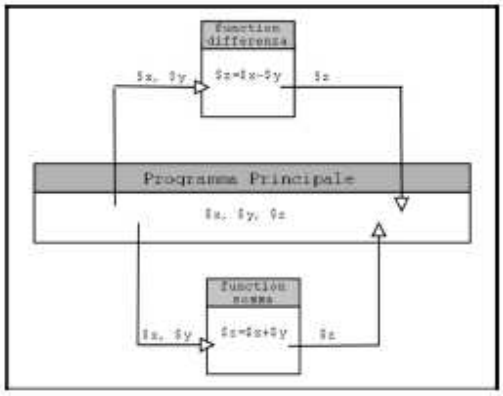
4b.8.1) Passaggio di variabili : metodo POST

Nel metodo '**post**' i dati non vengono visualizzati in nessun modo, essi sono spediti tramite il protocollo HTTP e non sono visibili sul navigatore.

Non c'è una regola particolare per scegliere l'uno piuttosto che l'altro metodo. Possiamo dire che se la quantità di dati da inviare è grande è preferibile utilizzare il '**post**'. Il protocollo http permette, tramite i form HTML, l'upload di file dall'elaboratore cliente a quello servente, in questi casi è bene utilizzare il metodo '**post**'. Nel file di configurazione del PHP ('`php.ini`') può essere definita sia la dimensione massima dei file che il servente accetterà tramite HTTP, sia la dimensione massima dei dati accettati tramite '**post**'. Quest'ultimo valore può essere definito valorizzando la variabile '**post_max_size**' nel '`php.ini`'.
Quando i dati da scambiare sono pochi e non importa che risultino direttamente visibili si può utilizzare il metodo '**get**', che è comodo anche per quelle situazioni in cui si utilizza spesso il tasto avanti e indietro del navigatore. Le variabili sono memorizzate insieme all'indirizzo e quindi ogni volta che si richiama una pagina ad essa vengono rifornite le variabili e i loro valori.

Dalle versioni del PHP 4.2 e successive la gestione delle variabili passate tramite moduli HTML con il metodo get, viene effettuata per mezzo della variabile predefinita ***$HTTP_GET_VARS.*** che è un array associativo (esempio `<?=$HTTP_GET_VARS["nome"]?>`). Per il metodo post si ha della variabile predefinita ***$HTTP_POST_VARS.*** che è un array associativo (esempio `<?=$HTTP_POST_VARS["nome"]?>`).

4b.9) Utilizzo di funzioni in PHP



Una funzione può essere intesa come un "sotto-programma" a cui si possono fornire dei valori per ottenere una risposta da essi dipendente o meno. Il PHP mette a disposizione degli sviluppatori una vasta gamma di funzioni predefinite, tuttavia in alcune situazioni è necessario eseguire operazioni particolari, in questi casi è possibile definire delle funzioni personali che si comportano secondo le proprie necessità.
Il manuale PHP riporta una lunga e dettagliata guida delle funzioni predefinite utilizzabili in PHP. Si rimanda al manuale l'uso e la spiegazione di ogni singola funzione.
Un modo molto semplice di schematizzare il concetto di funzione è pensare ad una scatola che permette tramite due fori di inserire ed estrarre dei valori. All'interno della scatola può essere utilizzato qualunque costrutto messo a disposizione dal linguaggio, ecco perchè si parla di "sottoprogrammi".
L'esempio classico è quello di una semplice funzione di saluto:

```
<?
//---------------- Definizione delle funzioni
function saluto($ora)f
/*
Questa funzione genera un messaggio di saluto
in base al valore della variabile $ora passatagli
che contiene l'ora, con valore tra 0 e 24
*/
if (5 < $ora && $ora <= 12){
      echo "Buon Giorno.";
  }elseif(
      <$ora && $ora <= 18){
      echo "Buon Pomeriggio.";
  } elseif(18<$ora && $ora <= 24){
      echo "Buona Sera!";
  }else {
      echo "Guarda che razza di ora è! Ma quando dormi?!";
  }
}
```

```
//---------------- Programma principale
$orario = date("H"); // estraggo l'ora attuale
// Richiamo la funzione e ci aggiungo del testo...
saluto($orario);
echo " Come va oggi?!?";
echo "<br><br> Ancora un saluto: "; saluto($orario);
?>
```

Esaminiamo la sintassi per la dichiarazione delle funzioni.

```
function nome_funzione(variabile_1, variabile_2, .... , variabile_n){
        istruzioni funzione
        return valore se necessario
}
```

Il costrutto '**`function`**' segna l'inizio della dichiarazione, subito dopo va assegnato un nome che deve essere unico nel programma, non possono esserci più funzioni aventi lo stesso nome, tra parentesi tonde è possibile elencare le variabili, se esistono, che devono essere fornite alla funzione, se sono più di una vanno separate da una virgola ('**`,`**'). Il corpo della funzione è, infine, contenuto tra parentesi graffe ('**`{...}`**'). La funzione può restituire dei valori, dunque quando richiamata può essere vista come una variabile contenente dei valori. Il comando '**`return`**' si occupa di riportare al programma chiamante il valore della variabile frutto del calcolo della funzione. Quando si richiama una funzione non importa il nome delle variabili che gli vengono fornite, quello che importa è l'ordine con cui esse vengono elencate. Dunque i valori passati alla funzione andranno nelle variabili locali (della funzione) definite al momento della dichiarazione della stessa.
Le funzioni sono dei piccoli programmi a se stanti, dunque le variabili utilizzate al loro interno, dette "variabili locali", non hanno lo stesso valore di quelle utilizzate nel programma principale, dette "variabili globali" anche se nominate allo stesso modo. Si può dire che sono variabili diverse. Un modo per far coincidere le variabili locali e globali è passare le variabili globali come argomento della funzione. Un secondo metodo, è l'utilizzo del comando '**`global`**' all'nterno della funzione. Questo permette di "catturare" il valore della variabile definita globale all'interno della funzione. Esempio:

```
function scheda($nome, $cognome){
        global $tel;
}
```

L'utilizzo di '**`global`**' non è consigliabile dal momento che ogni modifica alla variabile utilizzata come globale all'interno della funzione si ripercuote anche nel programma principale.

4b.9.1) funzioni ricorsive

Il PHP permette di richiamare le funzioni in modo ricorsivo, ovvero è possibile richiamare una funzione dal suo interno. Un classico esempio applicativo di questa tecnica è il calcolo del fattoriale. Come noto il fattoriale di un numero intero positivo ***n*** (indicato come ***n!***) è pari a ***n*(n-1)*(n-2)*...*(n-(n-1))***.
Ad esempio: ***5! = 5*4*3*2*1 = 120***
Ecco come risolvere il problema con l'utilizzo di una funzione ricorsiva:

```
<html><body>
<?
//---------------- Definizione delle funzioni
function fattoriale($numero)f
        if($numero <= 1){
                return 1; // abbandona la ricorsione
         }else{
                return $numero*fattoriale($numero-1); //la funzione richiama se stessa
        }
}
echo "<br>Calcolo del fattoriale:<br>";
for($i=-2; $i<=10; $i++){
        echo "<br>Il fattoriale di $i è: ".fattoriale($i);
}
?>
</body></html>
```

All'interno della dichiarazione, riga 9, viene richiamata la funzione stessa passandogli il numero decrementato di uno. Questo perchè ***n! = n*(n-1)! = n*(n-1)*(n-2)!*** e così via... Il fattoriale di 0 è pari ad 1 per definizione. Per semplificare il controllo si è stabilito che il fattoriale di un numero negativo è pari ad 1 anche se non è corretto matematicamente.

4b.10) La gestione del File system

In PHP si hanno molte funzioni per la gestione del sistema operativo: è possibile ottenere e cambiare i permessi, il proprietario e il gruppo di un file, creare collegamenti simbolici, copiare, spostare, eliminare directory e file. Prima di entrare nello specifico è bene precisare ancora una volta che il file system su cui il PHP opera è quello dell'elaboratore servente, quindi se si crea un file esso viene creato sul disco del servente e non su quello del cliente. Un file può essere visto come un contenitore nel quale immagazzinare informazioni. La gestione fisica del file su disco viene effettuata dal sistema operativo a cui si interfaccia il PHP. Per poter accedere alle informazioni contenute in un file è necessario aprirlo, leggerne il contenuto, eventualmente modificarlo e infine chiuderlo. I dati vengono salvati nel file sotto forma di righe, per indicare il fine riga nei file di testo viene utilizzato '**\n**', mentre la fine del file è ottenuta tramite la funzione '**feof()**' che restituisce un valore booleano vero se ci si trova alla fine del file. La particolarità fondamentale di questa struttura è il suo accesso, a differenza degli array non è possibile puntare direttamente ad una riga ben precisa del file; è dunque necessario scorrere il suo contenuto fino ad arrivare alla riga desiderata.

Se un file è un contenitore per poter accedere al suo contenuto è necessario aprirlo, per farlo si utilizza la funzione '**fopen()**' a cui va fornito il file con il percorso, se diverso da quello dello script, e il tipo di apertura. I tipi di aprtura ammessi sono:

- **'r'** Apre in sola lettura; posiziona il suo puntatore all'inizio del file;
- **'r+'** Apre in lettura e scrittura; posiziona il suo puntatore all'inizio del file;
- **'w'** Apre in sola scrittura; posiziona il suo puntatore all'inizio del file e ne elimina il contenuto. Se il file non esiste lo crea, vuotandolo;
- **'w+'** Apre in lettura e scrittura; posiziona il suo puntatore all'inizio del file e ne elimina il contenuto. Se il file non esiste lo crea, vuotandolo;
- **'a'** Apre in sola scrittura; posiziona il suo puntatore alla fine del file. Se il file non esiste lo crea;
- **'a+'** Apre in lettura e scrittura; posiziona il suo puntatore alla fine del file. Se il file non esiste lo crea;

Il PHP permette anche di aprire file remoti tramite i protocolli più diffusi come HTTP e FTP, ovviamente il file deve essere raggiungibile almeno in lettura.

Ecco qualche esempio di apertura di file:

```
$f = fopen ("dati.txt", "w");
$f = fopen ("/home/qualche_percorso/dati.txt", "a");
$f = fopen ("http://www.php.net/", "r");
$f = fopen ("ftp://nome_utente:password@indirizzo_del_sito_ftp.com/", "w");
```

Va notato che la funzione ritorna un intero. La variabile '**$f**' contiene i riferimenti al file aperto, viene anche detto "puntatore al file". Tutte le operazioni saranno eseguite sul puntatore al file.

La funzione '**file_exists()**', come riportato nel manuale ufficiale ritorna un valore booleano vero se il file esiste falso negli altri casi.

La funzione '**feof()**' resituisce valore vero quando si arriva alla fine del file.

Per la lettura delle singole righe del file è stata utilizzata la funzione '**fgets()**' che restituisce una stringa o un valore falso in caso di errore.

L'istruzione **@fclose($f)** chiude il puntatore al file definito nella fase di apertura. È bene liberare un file prima possibile per evitare problemi di scrittura concorrenziale.

Se il file di testo da leggere è piccolo si può utilizzare la funzione '**file("*nome_del_file*")**'. La funzione '**file()**' resituisce un array formato da tanti elementi quante sono le righe del file di testo, ogni elemento conterrà il testo della riga relativa. Ad esempio

```
$appo = @file("dati_1.txt");
if ($appo){
 echo "Operazione riuscita il contenuto del file dati_1.txt è nell'array <b>\$appo</b><br>";
 $i=0;
 foreach($appo as $valore) // visualizzo il contenuto dell'array
       echo "<br>".$valore;
}else{
  echo "Errore nella lettura del file! Accertarsi che esista e che i permessi siano corretti";
}
```

L'istruzione **@fputs($f,$frase)** scrive il testo presente in $frase, sul file $f.

L'istruzione **$var=@fgets($f,nbytes)** legge il testo (nbytes) presente in $f e lo assegna a $var.

4b.11) I cookies

Per motivi di riservatezza e soprattutto di sicurezza il protocollo HTTP non permette di scrivere sul disco dell'elaboratore cliente. Il linguaggio PHP consente di gestire i cookie, piccoli file di testo contenenti informazioni utili e non dannose per la gestione delle sessioni sul Web.
Per creare un cookie esiste la funzione **setcookie(nome, valore, espirazione, percorso, dominio, secure)** : **nome** è il nome del cookie, **valore** è il valore da assegnare al cookie, **espirazione** è la data di espirazione del cookie, percorso è la directory, a partire dal dominio per la quale il cookie è valido; dominio è il dominio per il quale il cookie è valido; secure è un valore che imposta se il cookie debba essere inviato tramite una connessione HTTPS. Come esempio l'istruzione:

```
setcookie("TEST", "Prova per il cookie test", time()+60, "/nomeutente", ".dominio.com", 0)
```

invia dalla nostra pagina un cookie chiamato TEST, con valore "Prova per il cookie test", con espirazione di un minuto dal momento dell'invio, per la directory "/nomeutente" del dominio http://www.dominio.com , senza utilizzare connessioni protette.
Il cookie va inviato prima di qualsiasi tag HTML.
I dati presenti nel cookie vengono inviati come un array di dati nella variabile $HTTP_COOKIE_VARS e possono essere letti per mezzo di un algoritmo del tipo:

```
if (isset($HTTP_COOKIE_VARS)){
        while(list($nome,$valore)=each($HTTP_COOKIE_VARS)){
                echo "$nome=$valore";
        }
}
```

4b.12) PHP e MYSQL

Mysql è un database relazionale open source, è molto veloce e professionale. Consente il salvataggio di grandi quantità di dati e l'accesso contemporaneo di molti utenti (101). Il PHP contiene al suo interno numerose funzioni per la connessione dei database Mysql; è per questo motivo che questa accoppiata sta avendo un successo enorme.
Installare Mysql è molto semplice, sia sotto win32 sia sotto Linux. E' possibile scaricare i pacchetti per questi due sistemi direttamente dal sito del mysql (www.mysql.com). E' anche possibile utilizzare il pacchetto 'easyphp' che installa direttamente sia il linguaggio PHP, sia il Web server apache sia Mysql.
Per comunicare e raccogliere i dati di un database Mysql PHP utilizza il linguaggio SQL (qui dato per conosciuto).

4b.12.1) Connessione a Mysql

La prima cosa da fare è la connessione al Mysql. A questo riguardo PHP ci fornisce una funzione apposita: **mysql_connect**. La sintassi della funzione di connessione è la seguente:

```
<?php
mysql_connect(nome_host,nome_utente,password);
?>
```

Il nome utente e la password sono indispensabili in quanto Mysql è un database molto sicuro e non si può accedere ad esso senza aver prima creato un accountPer quanto riguarda il nome dell'host esso è quasi sempre *localhost*. Vediamo uno script completo per la connessione:

```
<?php
// script per la connessione a mysql
$host = 'localhost';
$user = 'vostro_user';
$password = 'vostra_password';
mysql_connect($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// fine script di connessione
?>
```

La funzione "**die**" ci permette di stampare a video la scritta compresa tra virgolette nel caso la connessione non vada a buon fine.
Una volta connessi, per manipolare i nostri dati, dobbiamo selezionare il database da utilizzare. Se non abbiamo ancora creato nessun database dobbiamo prima crearlo e poi selezionarlo, utilizzeremo quindi varie funzioni PHP nel prossimo script:

```
<?php
// script per la connessione, creazione e selezione di un database Mysql
// Mi connetto a Mysql
$host = 'localhost';
$user = 'vostro_user';
$password = 'vostra_password';
mysql_connect($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// Creo il database "prova"
mysql_create_db("prova")or die ("Non riesco a creare il database");
// seleziono il database appena creato
mysql_select_db("prova") or die ("Non riesco a selezionare il database");
print "Connessione, creazione, selezione del database eseguita";
// Fine script
?>
```

Abbiamo visto due nuove funzioni PHP:

mysql_create_db("nome_database") **mysql_select_db**("nome_database");

Notate che in ogni funzione abbiamo inserito l'istruzione die, questo ci aiuta in caso di problemi ad identificare subito il perchè degli errori e la loro risoluzione.

4b.12.2) Tabelle e dati in Mysql

Prima di tutto si deve creare una tabella nella quale inserire i dati. Ogni tabella dovrà essere identificata da un nome univoco. Per quanto riguarda i campi, nella loro creazione bisogna specificare il tipo di dato che il campo dovrà contenere, Mysql supporta una moltitudine di tipologie dei campi a seconda dei dati che si dovranno inserire, vediamo quali sono i tipi più importanti. Per quanto riguarda i numeri reali Mysql supporta svariati tipi di archiviazione, la differenza fra i tipi elencati sotto non è nel formato del numero che andrà inserito nel campo ma la grandezza del numero (più grande è il numero è più è la memoria utilizzata):

TINYINT 1 byte
SMALLINT 2 byte
MEDIUMINT 3 byte
INT 4 byte
BIGINT 8 byte
FLOAT 4 byte
DOUBLE 8 byte

quanto riguarda le stringhe sono due i tipi di campo più utilizzati:

CHAR(numero_caratteri)
VARCHAR

Come possiamo intuire mentre in un campo CHAR la lunghezza è fissa e viene stabilita da noi durante la creazione della tabella; nel campo VARCHAR il numero di caratteri è variabile e non bisogna specificare preventivamente il numero di caratteri massimo che dovrà contenere il campo. Naturalmente i campi di tipo VARCHAR occuperanno più memoria e la lettura al loro interno impiegherà qualche decimo di secondo in più rispetto ai campi CHAR.

Esistono vari altri tipi di campo ma una loro trattazione esaustiva necessiterebbe di svariate pagine, tutta la documentazione sui tipi di dati supportati da Mysql è reperibile in italiano nel sito www.mysql.com.

Per capirci: è uno spreco di spazio e di prestazioni dichiarare un tipo di campo VARCHAR se sapete già che quel campo non potrà contenere un numero determinato di caratteri.

Supponiamo di voler costruire una tabella che raccolga una serie di indirizzi e-mail per una eventuale mailing-list del nostro sito. Dovremo costruire una tabella con 3 colonne: una colonna raccoglierà un numero identificativo che ci aiuti ad indicizzare i dati, una colonna per il nome e cognome degli utenti ed infine la colonna che conterrà le e-mail: Il codice sarà questo:

```
<?php
// script per la creazione di una tabella per la catalogazione delle e-mail per una mailing list
// per non appesantire il codice supponiamo di esserci già connessi
// al databese con le funzioni viste nelle lezioni precedenti
// scrivo l'istruzione SQL per la creazione della tabella
$sql= "CREATE TABLE mail (id_utente INT(10) NOT NULL,
        nome_cognome CHAR(100) NOT NULL,
        mail CHAR(50) NOT NULL)";
// Invio l'istruzione SQL a mysql
mysql_query("$sql") or die ("Non riesco a creare la tabella");
print " La tabella mail è stata creata con successo!";
// fine dello script
?>
```

In questo modo abbiamo creato la nostra tabella con tre colonne. La funzione usata per l'invio delle istruzioni SQL al database è **mysql_query**. La sua sintassi è molto semplice:

mysql_query("istruzioni SQL");

4b.12.3)Manipolazione dati

Ora che abbiamo creato la nostra tabella possiamo iniziare ad inserire i dati. Vediamo ora i comandi SQL e le funzioni PHP per inserire i dati.

```
<?php
// script per inserire i dati nella tabella mail per comodità supponiamo di esserci già connessi al database
// voglio inserire 1 nuovi indirizzi, scriveremo:
mysql_query("insert into mail (id_utente, nome_cognome,mail) values ('1','Mario Rossi','mario@suosito.com')");
// fine script
?>
```

Con questo piccolo script abbiamo inserito un nuovo indirizzo all'interno della tabella mail. Il formato dell'istruzione SQL Insert è:

INSERT INTO nome_tabella (nome_campi) values (dati_da_inserire_nei_campi);

Notate che l'elenco dei dati presente nella seconda coppia di parentesi tonde deve corrispondere all'ordine dei campi che abbiamo inserito nella coppia delle prime parentesi.

Ora che la nostra tabella contiene qualche dato possiamo leggere al suo interno per sapere quali dati ci sono al suo interno, per far questo utilizzeremo l'istruzione SQL "SELECT":

```
<?php
// script per leggere i dati contenuti in un campo della tabella  mail
// per comodità supponiamo di esserci già connessi al database
$dati = mysql_query("select * from mail");
$array = mysql_fetch_array($dati);
// fine script
?>
```

L'istruzione SELECT chiede i dati di una riga della tabella che abbiamo selezionato nell'istruzione FROM (nel nostro caso la tabella si chiama mail). Per poter utilizzare i dati che Mysql invia dobbiamo utilizzare la funzione **mysql_fetch_array** che crea un array associativo che ha come indice il nome delle colonne. Continuando lo script avremo che per visualizzare i dati che abbiamo estrapolato dal database scriveremo:

```
<?php
// codice per leggere i dati contenuti in un campo
print "Contenuto della colonna id_utente: $array['id_utente'] ";
print "Contenuto della colonna nome_cognome: $array['nome_cognome'] ";
print "Contenuto della colonna mail: $array['mail'] ";
?>
```

Naturalmente questo script legge una riga per volta e Mysql restituirà i dati dell'ultima riga inserita. Nel caso volessimo ottenere dei dati specifici di un utente dovremo usare l'istruzione WHERE:

```
<?php
// Volendo visualizzare i dati dell'utente Mario Rossi avrei scritto  in questo modo l'istruzione SQL
$dati = mysql_query("SELECT * FROM mail WHERE nome_cognome='Mario Rossi'");
$array = mysql_fetch_array($dati);
// fine script
?>
```

Questo script va bene nel caso dovessimo leggere solo una riga della tabella, nel caso in cui vogliamo invece leggere tutto il contenuto della tabella dovremo aggiungere un **ciclo while** così organizzato:

```
<?php
// script per leggere i dati contenutiin tutti i campi della tabella  mail
$dati = mysql_query("select * from mail");
while ( $array = mysql_fetch_array($dati)) {
   print "Contenuto della colonna id_utente: $array['id_utente'] ";
   print "Contenuto della colonna nome_cognome: $array['nome_cognome'] ";
   print "Contenuto della colonna mail: $array['mail'] ";
}
```

```
?>
```

Con questo script finchè la tabella mail non sarà vuota PHP creerà l'array associativo contenente i dati letti dal database.

4b.12.5) Manipolazione dei dati

Le ultime importanti nozioni sulla manipolazione dei dati inseriti nel database riguardano la modifica e la cancellazione dei dati. A questo scopo, rispetto agli script visti precedentemente riguardo l'inserimento cambiano solo le istruzioni SQL. Per la modifica utilizzeremo l'istruzione UPDATE mentre per la cancellazione l'istruzione DELETE.

```
<?php
// script per la modifica dei dati nella tabella mail
$dati = mysql_query ("UPDATE mail SET mail='mario@tiscalinet.it' WHERE nome_cognome='MARIO ROSSI'");
?>
```

L'istruzione UPDATE è molto semplice, ricordate sempre di specificare il WHERE perchè altrimenti la modifica verrà eseguita in tutti campi della tabella mail.
La sintassi dell'istruzione UPDATE:

```
UPDATE nome_tabella SET nome_colonna='nuovo_valore' WHERE nome_colonna='identificativo_colonna';
```

Nella creazione della tabella abbiamo all'inizio previsto una colonna che contiene l'dentificativo numerico del campo. Questo indice è importantissimo per le istruzioni di modifica e di cancellazione perchè in questo modo ogni riga ha un numero univoco e non si rischia di cancellare/modificare altre righe. Utilizzando l'identificativo avremo scritto:

```
<?php
// script per la modifica dei dati nella tabella mail
$dati = mysql_query ("UPDATE mail SET mail='mario@tiscalinet.it' WHERE id_utente='1'");
?>
```

L'istruzione DELETE permette di cancellare un intera riga dalla tabella. Utilizzate questa istruzione con molta cautela in quanto Mysql non chiede conferme, neanche per la cancellazione di grosse quantità di dati:

```
<?php
// script per la cancellazione di tutti i dati nella tabella mail
$dati = mysql_query ("DELETE FROM mail");
?>
```

Con questo script cancelliamo tutte le righe presenti all'interno della tabella mail. Nel caso volessimo cancellare una determinata riga inseriremo nell'istruzione SQL l'istruzione WHERE, come segue:

```
<?php
// script per la cancellazione di una riga nella tabella mail
$dati = mysql_query ("DELETE FROM mail where id_utente='1'");
?>
```

Questo script cancellerà la riga in cui l'id_utente è uguale a 1.

4b.12.6) Altre funzioni PHP Mysql

PHP contiene molte funzioni; segue un semplice elenco di quelle più importantanti:

mysql_num_rows()

Restituisce il numero di righe interessate dall'istruzione SQL:

```
<?php
// mysql_num_rows
$dati = mysql_query("SELECT * FROM mail");
$numero_righe = mysql_num_rows($dati);
?>
```

mysql_insert_id()

Restituisce l'ultimo id (se presente) della riga interessata dall' ultima operazione di INSERT:

```
<?php
// mysql_insert_id
$dati = mysql_query("INSERT INTO mail (id_utente, nome_cognome, mail values ('2', 'Mario Rossi',
mario@tiscalinet.it') ");
$ultimo_id = mysql_insert_id();
?>
```

mysql_drop_db

Elimina un database Mysql:

```
<?php
// mysql_drop_db
mysql_drop_db("nome_database_da_eliminare");
?>
```

mysql_list_dbs

Restituisce la lista dei database presenti nel server Mysql:

```
<?php
// mysql_list_dbs
$connessione = mysql_connect($host, $user, $password);
mysql_list_dbs("$connessione");
?>
```

mysql_list_tables

Restituisce la lista delle tabelle presenti nel database selezionato:

```
<?php
// mysql_list_tables
mysql_list_tables("nome_database");
?>
```

Questo l'elenco delle funzioni più usate, potete trovare l'elenco completo delle funzioni sul sito del PHP www.php.net

4b.12.7) Chiusura di una connessione Mysql

Una volta conclusa la fase di manipolazione dei dati è sempre opportuno chiudere tutte le connessioni al server Mysql e magari liberare la memoria occupata dai risultati della query SQL. E' buona norma specificare sempre queste due funzioni per non caricare inutilmente il server che sta eseguendo i vostri script e dare la possibilità a tutti gli utenti di poter accedere al database.

Com ultimo esempio vi fornisco uno script completo di connessione, manipolazione e disconnessione ad un server Mysql.

```
Php
// Mi connetto a Mysql
$host = 'localhost';
$user = 'vostro_user';
$password = 'vostra_password';
mysql_connect($host,$user,$password) or die ("Non riesco a connettermi");
print "Connessione eseguita";
// seleziono il database appena creato
mysql_select_db("mail") or die ("Non riesco a selezionare il database");
print "Il database è stato selezionato";
// visualizzo tutti i campi presenti nella tabella mail
```

```
$dati = mysql_query("select * from mail");
// inizio il ciclo while per la visualizzazione delle righe
while ($array = mysql_fetch_array($dati) {
  print "Contenuto colonna id_utente: $array[‘id_utente’] <br>";
  print "Contenuto colonna nome_cognome: $array[‘nome_cognome’] <br>";
  print "Contenuto colonna mail: $array[‘mail’]";
}
// libero la memoria occupata dall'istruzione SELECT
mysql_free_result($dati);
// chiudo la connessione al server Mysql
mysql_close();
// Fine script
?>
```

Da notare che la funzione **mysql_close** può non contenere argomenti in quanto Mysql chiuderà automaticamente tutte le connessioni aperte.