Il linguaggio PHP
Introduzione

Il nome **PHP**, acronimo per *Professional Home Pages*, già la dice lunga sulla sua vocazione per la Rete: lo scopo del linguaggio è quello di consentire agli sviluppatori *Web* di realizzare in modo veloce pagine dinamiche. La definizione ufficiale di **PHP**, per la quale condigliamo di visitare www.php.net, chiarisce le caratteristiche peculiari di questo linguaggio, e precisamente:

- il **PHP** è un linguaggio di *scripting*
- è un linguaggio **HTML**-embedded
- opera *server-side*, cioè lato *server*

Vediamo, uno per volta, il significato di questi punti. Il **PHP è un linguaggio di *scripting***. I programmi scritti in linguaggio **PHP**, denominati brevemente *script*, vengono eseguiti tramite un apposito *software*, l'interprete **PHP**. Quest'ultimo si occupa di leggere il codice **PHP** e, interpretandone le istruzioni, esegue le operazioni corrispondenti (ad esempio la lettura di un *file* o un calcolo aritmetico). Dunque il **PHP** è quello che tecnicamente si definisce un linguaggio interpretato ed in questo esso si differenzia da altri linguaggi di programmazione, come ad esempio *C++* e *Java*, il cui codice sorgente, per poter essere eseguito, deve prima essere compilato (tradotto cioè in codice macchina).

E' **HTML-embedded**. Questa caratteristica si riferisce al fatto che il codice **PHP** è immerso nell'**HTML**; gli *script* sono inseriti, in altre parole, nelle pagine **HTML** in cui devono produrre i loro effetti. Il *Web server* riconosce le pagine **PHP**, distinguendole da quelle statiche, sulla base dell'estensione, che non sarà la solita .htm o .html ma piuttosto .php3, .phtml o simile; quando il *server* riconosce una estensione associata a **PHP** passa il testimone all'interprete, lasciando che sia quest'ultimo ad occuparsene (come descritto al punto precedente).

Opera server-side. Ovvero: il **PHP** opera lato *server*. Di conseguenza, chi accede ad una pagina **PHP** non ha la possibilità di leggere le istruzioni in essa contenute: essendo state processate ciò che il *client* vedrà sarà il risultato dell'elaborazione; per riassumere in uno slogan, insomma, il *client* vedrà cosa fa lo *script* ma non come lo fa.

Il Primo programma

Il primo esempio che vedremo sarà il classico messaggio di saluto Ciao mondo e ci servirà per mostrare la sintassi da utilizzare per includere codice **PHP** in una pagina *Web*. Ecco il nostro sorgente:

```
<html>
<head><title>Esempio 1</title></head>
<body>

<?php
echo "<h1>Ciao mondo!</h1>";
?>

</body>
</html>
```

Come si vede, si tratta di una normale pagina **HTML** in cui compaiono speciali marcatori che denotano l'inizio e la fine di un blocco di istruzioni **PHP**. Nell'esempio mostrato viene utilizzata la sintassi classica: l'inizio del codice viene contrassegnato con <?php mentre con ?> se ne indica la fine. Il risultato che otterremo, e cioè la pagina che verrà inviata al *browser*, sarà il seguente.

```
<html>
<head><title>Esempio 1</title></head>
<body>

<h1>Ciao mondo!</h1>

</body>
</html>
```

Si nota immediatamente che non vi è nessuna traccia del codice originario! In altri termini, il *client* non ha alcun modo per risalire alle istruzioni **PHP** che hanno generato la pagina richiesta.

Tornando alla sintassi per l'immersione di codice nell'**HTML**, ne esistono altre varianti; lo stesso blocco di istruzioni del primo esempio può essere scritto, in modo del tutto equivalente, così (sintassi abbreviata):

```
<?
echo "<h1>Ciao mondo!</h1>";
?>
```

così (sintassi in stile *Microsoft* **ASP**):

```
<%
echo "<h1>Ciao mondo!</h1>";
%>
```

o, ancora, così:

```
<script language="php">
echo "<h1>Ciao mondo!</h1>";
</script>
```

Quest'ultima forma può essere particolarmente conveniente se si utilizzano degli *editor* **HTML** visuali (come ad esempio *Front Page*) che potrebbero non tollerare gli altri tipi di sintassi.

Funzioni di base

La prima funzione di cui ci occupiamo è **phpinfo()**. Quello che fa è generare dinamicamente una (lunga) pagina contenente moltissime informazioni sulla versione di **PHP** installata e sull'ambiente di esecuzione. Per richiamare **phpinfo()** é sufficiente uno *script* come il seguente:

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

La pagina *Web* generata da `phpinfo()` consente di visualizzare la configurazione della nostra installazione di **PHP**, di conoscere quali estensioni sono disponibili e, cosa particolarmente importante per un neofita, di imparare i nomi delle variabili predefinite che **PHP** mette a disposizione del programmatore. La seconda funzione di cui facciamo la conoscenza è certamente la più utilizzata in ogni *script* **PHP**: `echo()`. La funzione `echo()` serve per scrivere (o stampare) l'*output* che viene inviato al *browser* del visitatore che accede al nostro *script*. Si consideri il

seguente esempio:

```
<html>
<head><title>echo</title></head>
<body>
<?php
echo "<h1>Benvenuto!</h1>";
?>
</body>
</html>
```

La pagina che verrà inviata al *client*, dopo l'elaborazione da parte dell'interprete **PHP**, sarà la seguente:

```
<html>
<head><title>echo</title></head>
<body>
<h1>Benvenuto!</h1>
</body>
</html>
```

Grazie ad **echo()** possiamo visualizzare il contenuto di variabili; nell'esempio seguente viene mostrato, nel messaggio di benvenuto, anche il nome di dominio del sito su cui lo *script* viene eseguito (nome contenuto nella variabile `$HTTP_HOST`).

```
<html>
<head><title>echo</title></head>
<body>
<?php
echo "<h1>Benvenuto su $HTTP_HOST!</h1>";
?>
</body>
</html>
```

Se, ad esempio, lo *script* viene eseguito sul sito www.latoserver.it, la pagina risultante sarà...

```
<html>
<head><title>echo</title></head>
<body>
<h1>Benvenuto su www.latoserver.it!</h1>
</body>
</html>
```

A rigore occorre rilevare che *echo* non è propriamente una funzione bensì un costrutto del linguaggio; per questo motivo non è necessario, come si è visto negli esempi, utilizzare le parentesi tonde per racchiudere gli argomenti.

`exit() e die()`

entrambe producono il risultato di arrestare l'esecuzione dello *script*, con la differenza che `die()` consente anche di stampare un messaggio. Ad esempio il seguente *script*...

```
<html>
<head><title>exit</title></head>
<body>
<? exit(); ?>
<p>Questa frase non si vedrà</p>
</body>
```

```
</html>
```

produce questo *output*:

```
<html>
<head><title>exit</title></head>
<body>
```

Le funzioni `exit()` e `die()` possono essere utilizzate, quindi, per gestire eventuali situazioni di errore che non consentono di proseguire l'esecuzione del nostro *script* **PHP** (i cosiddetti errori fatali). In queste circostanze può essere conveniente usare `die()` per mostrare un messaggio di errore appropriato.

Vediamo un semplice esempio. Nello *script* seguente controlliamo il valore della variabile globale $n e mostriamo un messaggio di errore, bloccando l'esecuzione del programma, se questo è maggiore di 1.

```
<html>
<head><title>die</title></head>
<body>

<?
$n = 5;
if ($n > 1) die("<h1>\$n è maggiore di uno!!!</h1>");
?>

<h1>$n è minore o ugale ad uno!</h1>

</body>
</html>
```

Il risultato sarà il seguente:

```
<html>
<head><title>die</title></head>
<body>

<h1>$n è maggiore di uno!!!</h1>
```

Se, invece, sostituiamo l'istruzione $n=5 con $n=1 il risultato diventa:

```
<html>
<head><title>die</title></head>
<body>

<h1>$n è minore o ugale ad uno!</h1>

</body>
</html>
```

Variabili

Per creare una variabile è sufficiente assegnarle un valore; in altre parole non vi è alcuna necessità di dichiararla esplicitamente, come avviene, invece, in altri linguaggi. Si parla, dunque, di dichiarazione implicita. Se vogliamo visualizzare nella nostra pagina **PHP** il valore di una variabile, in modo che

venga mostrata dal *browser* del visitatore, possiamo usare la funzione *echo*. Ad esempio:

```
// Questa istruzione visualizza il valore della
// variabile $a
echo $a;
```

A disposizione del programmatore c'è anche un certo numero di variabili predefinite, cioè di variabili il cui valore è già impostato. Solitamente queste variabili contengono informazioni circa l'ambiente di esecuzione dello *script* **PHP**. Esaminiamone alcune di uso frequente.

- **$PHP_SELF**. La variabile $PHP_SELF contiene il percorso dello *script* in esecuzione; ad esempio, all'interno dello *script* **PHP** raggiungibile all'indirizzo

  http://www.latoserver.it/index.php3

  il valore della variabile $**PHP**_SELF sarà /index.php3.

- **$HTTP_HOST**. La variabile `$HTTP_HOST` contiene il nome del *server* su cui lo *script* viene eseguito; nel caso dell'esempio precedente il valore di `$HTTP_HOST` sarebbe www.latoserver.it.
- **$HTTP_REMOTE_HOST e $HTTP_REMOTE_ADDR**. Le variabili `$HTTP_REMOTE_HOST` e `$HTTP_REMOTE_ADDR`, invece, forniscono rispettivamente il nome di dominio e l'indirizzo *IP* del visitatore.

In uno *script* **PHP** i nomi di variabili sono prefissati dal simbolo $ (dollaro); ad esempio, la seguente istruzione:

```
$a = $b + $c
```

assegna ad una variabile di nome a la somma dei valori di altre due variabili, rispettivamente b e c. Il simbolo = è l'operatore di assegnamento.

I Tipi di Dato

Vediamo adesso cosa può contenere una variabile. Abbiamo parlato di valori, ma a che tipo di valori facciamo riferimento? Introduciamo dunque i tipi di dato che **PHP** mette a disposizione del programmatore.

I tipi di dato supportati da **PHP** si distinguono in tipi scalari e tipi compositi: sono tipi scalari i numeri (sia interi che in virgola mobile) e le stringhe; sono tipi compositi gli *array* e gli oggetti. A partire dalla versione 4 di **PHP**, inoltre, è stato introdotto un nuovo tipo scalare: quello booleano. Una variabile booleana può assumere solo due valori, vero (costante `TRUE`) o falso (costante `FALSE`).

```
// $b è una variabile di tipo bool
$b = TRUE;

// $num è una variabile di tipo intero
$num = 25;

// $pi_greco è un numero in virgola mobile;
// si noti il punto (virgola decimale)
$pi_greco = 3.14;

// $messaggio è una stringa
$messaggio = "Ciao a tutti!";
```

Un *array* in **PHP** può corrispondere sia ad un vettore, cioè ad una struttura dati in cui ogni elemento è individuato da un indice numerico, sia ad una tabella di *hash*, cioè (in modo molto semplice) ad una collezione di coppie nome/valore. In **PHP**, infatti, tali strutture sono sostanzialmente la stessa cosa. Un *array* può essere creato esplicitamente utilizzando il costrutto `array()` oppure implicitamente. Vediamo alcuni esempi.

```
// Questo è un array di numeri interi
// Lo creo esplicitamente usando array()
$primi = array( 2, 3, 5, 7, 11 );

// Questo array lo creo implicitamente
$pari[0] = 2;
$pari[1] = 4;
$pari[2] = 6;

// Questa è una tabella di hash
$bookmark["puntoinf"] = "punto-informatico.it"
$bookmark["latoserver"] = "www.latoserver.it"
```

Per accedere ad un elemento di un *array* si può utilizzare sia il corrispondente indice numerico, sia la chiave (se si tratta di una tabella di *hash*). Ad esempio:

```
// Questa istruzione stampa "7"
// cioé l'elemento di indice 3 dell'array $primi
echo $primi[3];

// Questa istruzione stampa "www.latoserver.it"
echo $bookmark["latoserver"];
```

A differenza di quanto avviene in altri linguaggi di programmazione, un *array* in **PHP** può contenere elementi di tipo diverso. Ad esempio, è perfettamente legittimo costruire un *array* che contenga sia numeri interi che stringhe.

```
// Questo è un array valido!
// Contiene: un numero intero, una stringa,
// un numero in virgola mobile ed un altro array!
$mix = array( 1, "ciao", 3.14, array( 1, 2, 3 ) );
```

Il tipo di dato *object* verrà trattato in una lezione successiva dedicata alla programmazione orientata agli oggetti con **PHP**

I metodi GET e POST

Come è stato discusso all'inizio del modulo 15, una *form* **HTML** utilizza i metodi *GET* o *POST* per trasferire informazioni a/dal *server*. In alternativa, si può accedere alle informazioni inviate con i metodi *GET* e *POST* utilizzando gli *array* associativi **$HTTP_GET_VARS** e **$HTTP_POST_VARS**, rispettivamente. Ad esempio, se tramite il metodo *GET* inviamo ad uno *script* **PHP** un parametro di nome pagina e di valore pari a 1, all'interno dello *script* stesso potremo accedere a tale informazione sia usando la variabile globale `$p`, sia accedendo a `$HTTP_GET_VARS ["p"]`, cioè all'elemento dell'array associativo `$HTTP_GET_VARS` individuato dalla chiave p.

Supponiamo di scrivere nella casella di testo la parola Schumacher e di premere il pulsante Invia i dati. Il *browser* si sposterà all'indirizzo

http://www.pippo.it/scripts/elabora.php?campione=Schumacher

Cosa è successo? Poiché la *form* dell'esempio usa il metodo *GET* per trasferire i propri dati, questi vengono codificati nell'indirizzo dello *script* a cui devono essere inviati. Nello *script* elabora.php adesso troveremo definita una variabile globale di nome `$campione` il cui valore è la stringa Schumacher.

```
// Nel file `elabora.php' ...
// Questo stampa "Schumacher"
echo $campione;
```

A questo punto è utile accennare al modo in cui le informazioni provenienti da una *form* **HTML** vengono codificate per essere trasmesse con il metodo *GET*. Partiamo dall'indirizzo URL dello *script* a cui vogliamo inviare tali dati; ad esso aggiungiamo (a destra) un punto interrogativo che separerà la URL vera e propria (a sinistra) dalla query string che andiamo a costruire.

La struttura della *query string* consiste di una serie di coppie nome/valore separate da una e commerciale (&); i caratteri non ammissibili in un indirizzo URL (ad esempio gli spazi) vengono sostituiti dal simbolo di percentuale seguito dal corrispondente codice *ASCII* (in esadecimale).

Adesso che sappiamo come funziona il metodo *GET* possiamo sfruttarlo per passare parametri ad uno *script* **PHP**. Supponiamo di avere uno *script* di nome news.php che estrae notizie ed articoli da un *database*; ad esso vogliamo passare un parametro, chiamiamolo `$argomento`, che determinerà il tipo di notizie che ci verranno mostrate. Ad esempio, per ottenere notizie sportive, invocheremo:

http://www.pippo.it/news.php?argomento=Sport

In questo caso la codifica manuale della URL era semplice, ma cosa fare se ci interessano le notizie di Attualità e Cultura? Niente paura, esiste una funzione **PHP**, `urlencode()`, che ci permette di codificare una stringa in una forma ammissibile per una URL. Il frammento di *script* per creare il *link* appropriato sarà:

```
<?
echo '<a href="http://www.pippo.it/news.php?argomento=';
echo urlencode("Attualità e Cultura");
echo '">Clicca qui</a>';
?>
```

I Cookies

Vediamo adesso come si manipolano i *cookies*, introdotti nel modulo 15, con il linguaggio **PHP**. Tutte le operazioni di scrittura, modifica o cancellazione di *cookies* in **PHP** avvengono mediante una stessa funzione, `setcookie()`. Tale funzione deve obbligatoriamente essere invocata prima che qualsiasi contenuto venga inviato al *browser*; i *cookies*, infatti, vengono trasmessi tra *client* e *server* sotto forma di intestazioni (*headers*) HTTP.

La funzione prevede solo due argomenti obbligatori: il nome da assegnare al *cookie* ed il suo valore. Ad esempio, se vogliamo memorizzare nel *browser* di un visitatore un *cookie*, che chiameremo $nomeutente, contenente la stringa "latoserver", l'istruzione da utilizzare sarà la seguente

```
// Imposto un cookie: $nomeutente = "pippo.it";
setcookie( "nomeutente", "pippo.it" );
```

E' possibile specificare anche altri argomenti (facoltativi); nell'ordine abbiamo: scadenza del *cookie*, percorso, dominio e *secure*. Per una discussione dettagliata si rinvia alla sitografia in fondo alla pagina.

Quando un *cookie* è stato impostato, il suo valore può essere modificato richiamando nuovamente la funzione `setcookie()` ed associando allo stesso nome il nuovo valore. La cancellazione di un *cookie*, infine, può avvenire in due modi: assegnandogli un valore nullo o impostando la scadenza ad una data passata.

Resta da vedere in quale modo si possano leggere da uno *script* **PHP** le informazioni memorizzate nei *cookies*; in questo compito siamo molto facilitati, in quanto l'interprete **PHP** analizza automaticamente i *cookies* inviati dal *browser* e li rende disponibili in altrettante variabili globali e nell'*array* associativo `$HTTP_COOKIE_VARS`.

Passiamo ad analizzare un esempio pratico di utilizzo dei *cookies*, realizzando una semplice pagina **PHP** che ricorda la *data* e l'ora dell'ultimo accesso del visitatore. A tale scopo impostiamo sul *browser* un *cookie* `$ultimavisita`, la cui scadenza determina quanto a lungo viene conservata tale memoria. Nel nostro esempio non è stata specificata alcuna scadenza, in modo da far scomparire il *cookie* alla chiusura del *browser*. Il valore assegnato di volta in volta al *cookie* `$ultimavisita` è il risultato della funzione `time()` e consiste nel numero di secondi trascorsi dalla cosiddetta Unix epoch (primo gennaio 1970).

```
<?php
// file `saluto.php'
// Il saluto predefinito
$saluto = "Benvenuto!";

// Controllo se esiste il cookie...
if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
// Cambio il saluto con uno piu' appropriato
$saluto = "Bentornato!";
}

// Imposto il cookie relativo a questa visita
setcookie( "ultimavisita", time() );
?>
<html>
<head>
<title><? echo $saluto ?></title>
</head>
<body>
<h1><? echo $saluto ?></h1>

<?php if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
// Stampo la data dell'ultima visita
echo "L'ultima volta sei stato qui il " . date( "d/m/Y");
echo " alle ore " . date( "H:i:s.", $ultimavisita );
// Link per cancellare il cookie
echo "<p><a href=\"cancella.php\">Cancella il cookie</a>";
} else {
echo "Non sei mai stato qui prima?";
}
?>
</body>
</html>
```

Lo *script* cancella.php, richiamabile cliccando sul *link* Cancella il cookie, non fa altro che cancellare il *cookie* `$ultimavisita`, assegnandogli un valore nullo, e dirottare il *browser* di nuovo alla pagina precedente.

```
<?
// file `cancella.php'
```

```
setcookie( "ultimavisita", "" );
header( "Location: saluto.php" );
?>
```

L'esempio proposto, sebbene di improbabile utilità, dovrebbe aver chiarito le modalità d'impiego dei cookies; pur nella sua semplicità, inoltre, può essere il punto di partenza per lo sviluppo di funzionalità personalizzate, lasciate alla vostra fantasia.

L'accesso ai database

La coppia **PHP** e *MySQL* è sicuramente una delle più diffuse nella realizzazione di applicazioni *Web* basate su *software OpenSource*. *MySQL* è un *DBMS*, *Data Base Management System*, cioè un *software* per la gestione di basi di dati; la sua popolarità è indiscussa, grazie alle prestazioni di tutto rispetto e nonostante la mancanza di diverse caratteristiche avanzate (transazioni, *stored procedures*, eccetera).

Nel seguito vedremo in che modo è possibile, da uno *script* **PHP**, collegarsi ad un *database MySQL* ed eseguire operazioni su di esso tramite il linguaggio *SQL*. Per una completa comprensione della lezione è necessaria una conoscenza di base di *SQL*; assumeremo, inoltre, che *MySQL* sia correttamente installato.

L'accesso ad un *database MySQL* avviene mediante autenticazione; questo vuol dire che, prima di poter effettuare qualsiasi operazione su di esso, dobbiamo disporre dei privilegi necessari. In particolare, le informazioni di cui abbiamo bisogno sono: il nome dell'*host* su cui è in esecuzione il *server MySQL*, il nome del nostro *database*, il nome utente che ci è stato assegnato e la relativa *password* (per averle occorre rivolgersi all'amministratore di sistema). Supponiamo che i parametri nel nostro caso siano i seguenti:

```
// Il nome dell'host (hostname) su cui si trova MySQL
$dbhost = "localhost";

// Il nome del nostro database
$dbname = "dbprova";

// Il nostro nome utente (username)
$dbuser = "luca";

// La nostra password
$dbpass = "secret";
```

E' opportuno salvare tali parametri in apposite variabili piuttosto che utilizzarli direttamente nelle chiamate di funzione; in tal modo, infatti, potremo inserirli in un *file* separato che includeremo in tutti gli *script* che accedono al *database*.

La prima funzione che utilizziamo è *mysql_connect()*, che ci servirà per instaurare la connessione con il *server MySQL*. I parametri da fornire a tale funzione sono il nome dell'*host*, il nome utente e la *password*. Vediamo un esempio:

```
// Funzione mysql_connect()
$conn = mysql_connect($dbhost,$dbuser,$dbpass)
or die("Impossibile collegarsi al server MySQL.");
```

Si osservi la sintassi utilizzata nell'esempio precedente; il significato è il seguente: se la funzione restituisce un valore nullo, situazione che denota l'impossibilità a collegarsi al *server MySQL*, viene invocata `die()` per arrestare l'esecuzione e visualizzare un messaggio di errore. Inoltre, in una

variabile che abbiamo chiamato `$conn` salviamo il valore restituito da `mysql_connect()`, che servirà da identificativo della connessione stabilita.

Il passo successivo è la selezione del *database* su cui operare; la funzione da utilizzare è `mysql_select_db()`, alla quale occorre fornire il nome del *database* e, opzionalmente, l'identificativo della connessione (se non viene indicata verrà utilizzata l'ultima aperta).

```
// Funzione mysql_select_db()
mysql_select_db($dbname,$conn)
or die("Impossibile selezionare il database $dbname");
```

Opinioni e commenti

Scrivi nuovo

Anche questa volta in caso di insuccesso viene arrestata l'esecuzione del programma **PHP** e viene visualizzato un messaggio di errore appropriato.

Arriviamo così alla parte più importante, quella dell'interazione con la base di dati, interazione che avverrà mediante il linguaggio *SQL*. Le funzioni **PHP** che utilizzeremo in questa fase sono `mysql_query()`, per l'esecuzione di comandi *SQL*, e `mysql_fetch_row()`, per prelevare il risultato restituito da *MySQL* quando il comando eseguito è un'istruzione *SELECT* (quindi un'interrogazione, o *query*).

Supponiamo di voler creare una tabella del *database* con cui gestire una rudimentale rubrica telefonica. Il comando *SQL* da utilizzare sarà del tipo

```
CREATE TABLE rubrica(
Progressivo int PRIMARY KEY AUTO INCREMENT,
Nome varchar(40),
Cognome varchar(40),
Telefono varchar(20))
```

Per eseguire tale comando dal nostro *script* **PHP**, lo inseriamo dapprima in una stringa, ad esempio nel modo seguente

```
$sql = "CREATE TABLE rubrica( "
. "Progressivo int PRIMARY KEY AUTO INCREMENT, "
. " Nome varchar(40), Cognome varchar(40), Telefono varchar(20))";
```

Passiamo poi all'esecuzione vera e propria, invocando la funzione `mysql_query()`.

```
// Esegue il comando SQL o stampa un messaggio di errore
$res = mysql_query($sql,$conn)
or die( "Errore: " . mysql_error() );
```

Voilà, il gioco è fatto! In caso di errori, comunque, l'istruzione `mysql_error()` ci fornisce la descrizione del problema che si è verificato. Esaminiamo adesso il caso di una interrogazione del *database*. L'esecuzione del relativo comando *SQL* (sarà, naturalmente, una *SELECT*) è del tutto analoga al caso già visto. L'unica differenza risiede nel fatto che, in questo caso, abbiamo un risultato da prelevare. Una delle funzioni che possiamo utilizzare a tale scopo è `mysql_fetch_row()`.

```
// Interroghiamo la nostra rubrica

// Comando SQL da eseguire
```

```
$sql = "SELECT Telefono FROM rubrica "
. "WHERE Nome='Luca' AND Cognome='Balzerani'";

// Esecuzione comando SQL o messaggio di errore
$res = mysql_query($sql,$conn)
or die( "Errore: " . mysql_error() );

// Estrazione del risultato
$info = mysql_fetch_row($res);
echo "Il mio numero di telefono è " . $info[0];
```

Al termine della sessione di lavoro si può invocare la funzione `mysql_close()` per chiudere la connessione con il *server MySQL*. Si tratta, in ogni caso, di un passo opzionale in quanto tutte le connessioni lasciate aperte verranno chiuse automaticamente alla fine dello *script*.

```
// Funzione mysql_close()
mysql_close($conn);
```