**CORSO PHP**

 **A cura dell’Ing. Buttolo Marco**

PHP è l’acronimo di **Hypertext Preprocessor** ed è un potente linguaggio di scripting molto usato su web. Gli script scritti in PHP sono file con estensione ‘.php’. PHP è un linguaggio con una sintassi di tipo C-like, e pertanto chi conosce il C è molto avvantaggiato nell’apprendimento di PHP. Ad ogni modo non è un linguaggio particolarmente complicato e come si vedrà tra breve è molto potente. PHP è un **linguaggio lato server** ossia gli script PHP risiedono su un web server e vengono richiamati dal clienti (dalle pagine HTML) quando l’utente deve fare alcune particolari elaborazioni. Per esempio, se l’utente visualizza su proprio browser una pagina HTML di registrazione dei propri dati personali,una volta che tali dati vengono inseriti, vengono inviati ad uno script PHP residente su server il quale effettua la registrazione di tali dati su database presente su server. Graficamente si ha una cosa del seguente tipo:

WEB SERVER CON SCRIPT PHP.

CLIENT (WEB BROWSER)

 request

 response

 Figura 1

Un web server su cui fare esperimenti, e liberamente scaricabile, è il server **Wamp** (per chi ha Windows) **Lamp** (per chi ha Linux). Tale server include Apache (probabilmente il web server più famoso), l’interprete PHP, e MySQL. Pertanto, Wamp contiene tutto l’occorrente per incominciare. Vediamo subito un primo esempio chiarificatore.

<?php

 echo "test!!";

 ?>

Tale file, che per comodità ho chiamato test.php va inserito sotto la **document root** del server Wamp (es: c:\wamp\www). Quindi prima di avviare l’esempio bisogna avviare il server wamp cliccando sulla relativa iconcina presente su Desktop (che è stata inserita dal programma di installazione).



 Figura 2

Il risultato grafico è il seguente:



 Figura 3

Il risultato non è nulla di che, ma è pur sempre uno script PHP perfettamente funzionante. Solitamente i frammenti di codice PHP vengono inseriti direttamente all’interno di codice HTML, come viene mostrato di <html>

<head>

 <title>

 Test

 </title>

</head>

<body>

 <?php

 echo "Test!";

 ?>

</body>

</html>

Come si può facilmente notare il codice PHP inizia con un tag di apertura **<?php** e con un tag di chiusura **?>**.

Vediamo un altro esempio banale:

<?php

 $x=2;

 echo $x;

 ?>



 Figura 4

L’istruzione **echo** mostra a video il messaggio racchiuso tra doppi apici. Ogni istruzione in PHP termina con un punto e virgola. Le variabili in PHP vengono utilizzate con il simbolo del $ davanti al nome. PHP non è tipizzato, ed è case-sensitive ossia la variabile di nome x è differente dalla variabile di nome X.

Vediamo ora un altro esempio:

<?php

 $x=2;

 $y=3;

 echo $x, "<br>";

 echo $y;

 ?>

Il tag HTML <BR> permette il ritorno a capo.



 Figura 5

Vediamo un esempio un un pelino più complesso:

<?php

 $a = 10;

 $b = 11;

 $risultato = $a + $b;

 echo $risulato;

 ?>

Chiaramente il codice non ha bisogno di ulteriori commenti. A proposito di commenti, i commenti in PHP vengono inseriti nel seguente modo:

//questo è un commento.

Vediamo un ulteriore esempio un po’ più complicato:

<?php

 $numeri = array('1', '2', '3', '4', '5');

 //ciclo for scansione vettore

 for($i=0;$i<count($numeri);$i++)

 {

 echo $numeri[$i];

 }

 ?>

In questo esempio viene mostrato l’uso del classico ciclo **for**. Tale ciclo è così strutturato:

 for(condizione iniziale; condizione di uscita ciclo; incremento contatore)

 lista di istruzioni.

Si noti l’uso delle parentesi graffe. Se all’interno di un ciclo è presente una sola istruzione l’uso delle parentesi graffe non è obbligatorio ma è consigliabile in quanto rende il codice più leggibile. Se le istruzioni sono più numerose l’uso delle parentesi graffe è obbligatorio. In PHP la dichiarazione di un vettore avviene tramite la parola chiave **array**, come viene chiaramente mostrato nell’esempio. Si presti particolare attenzione al fatto che gli script PHP hanno un limite di tempo per la loro esecuzione, oltre il quale si arrestano. Tale limite è normalmente di 30 secondi, ed è comunque impostabile attraverso il file php.ini (il file di inizializzazione). Vediamo un successivo esempio sempre più complesso:

<?php

 $x=2;

 if ($x==2)

 echo $x;

 else

 echo "x non vale 2";

 ?>

Questo semplice esempio di codice PHP mostra l’uso del costrutto **if**. In sostanza se la condizione all’interno dell’if è vera si esegue l’istruzione contenuta al suo interno (echo $x), altrimenti se è falsa si esegue un’altra istruzione (echo “x non vale 2”). Pertanto in linea generale si ha:

 if(condizione)

 azione 1

 else

 azione2

Viene eseguita l’azione 1 soltanto se la condizione è verificata, altrimenti viene eseguita l’azione 2.

<?php

 $x = 1;

 while ($x <= 10)

 {

 $ris = 5 \* $x;

 print("5 \* $x = $ris<br>");

 $x++;

 }

 ?>

In questo esempio viene mostrato l’uso del ciclo **while**. Fintanto che la condizione del ciclo è verificata vengono eseguite le istruzioni al suo interno. L’incremento di un’unità del valore di una variabile si ottiene tramite ++. Analogamente si ha per il ciclo **do while**.

<?php

 $x = 4;

 do

 {

 $ris = 5 \* $x;

 print("5 \* $x = $ris<br>");

 $x++;

 } while ($x <= 10)

 ?>

Si noti l’uso dell’istruzione **print** che permette di visualizzare a video messaggi e valori di variabile (come l’istruzione echo). La differenza tra echo e print sta nel fatto che print permette di formattare le variabili inserite nella stringa, mentre l’istruzione echo no. Supponiamo ora di avere a disposizione due stringhe. Vediamo come si effettua il concatenamento di stringhe:

<?php

 $str1="Marco";

 $str2=" Buttolo";

 $str = $str1.$str2;

 print($str);

 ?>

L’output è il seguente:



 Figura 6

Pertanto l’operatore ‘.’ Permette la concatenazione tra due stringhe. Nei precedenti esempi abbiamo visto altri operatori come per esemio la somma (+). Esistono chiaramente anche la differenza (-), la moltiplicazione (\*), la divisione (/), e così via. Per incrementare il valore di una variabile di un’unità è possibile seguire due strade:

 $variabile=$variabile+1;

oppure:

 $variabile++;

Analogamente si ha per la sottrazione di un’unità:

 $variabile=$variabile-1;

 $variabile--;

Per verificare se il valore di due variabili è uguale si usa l’operatore ==, mentre per verificare se tali valori sono diversi si usa !=. Si noti la somiglianza sintattica che c’è tra il linguaggio PHP ed il linguaggio C. Infatti PHP è unlinguaggio C-like.

Anche in PHP ci sono gli **operatori logici**. Vediamoli brevemente:

* AND (&&)
* OR (||)
* NOT (!)

Vediamo subito un esempio:

<?php

 $x=1;

 $y=2;

 if ($x==1 && $y==2)

 echo "Entrambi i valori sono giusti!!";

 else if ($x==1 || $y==2)

 echo "Almeno un valore corretto!!!";

 ?>

Come mostra l’esempio viene eseguito il primo echo solo se entrambe le condizioni sono vere (l’AND funziona così), mentre viene eseguita il secondo echo solo se almeno una condizione è verificata.

Supponiamo ora di avere un ciclo, e di voler uscire da tale ciclo alla quinta iterazione. L’istruzione per effettuare ciò è l’istruzione **break**, come viene mostrato di seguito:

<?php

 for($index=0;$index<10;$index++)

 {

 if($index==4)

 break;

 }

 ?>

L’istruzione invece che permette di saltare alla successiva iterazione è l’istruzione **continue** che viene mostrata nel seguente esempio:

<?php

 for($index=0;$index<10;$index++)

 {

 if($index==4)

 continue;

 }

 ?>

Anche il PHP5, come molti altri linguaggi è procedurale, ossia è possibile utilizzare delle funzioni, opportunamente richiamate, che svolgono alcuni compiti specifici. Tali funzioni possono essere incorporate nel linguaggio (quelle predefinite) oppure possono essere create dall’utente. Per richiamare una particolare funzione basta scriverne il nome con gli eventuali parametri:

 nome\_funzione(par1, par2,…)

Vediamo brevemente alcune funzioni elementari predefinite. Analizziamo la funzione **empty** la quale verifica se una variabile è vuota o meno (se èdefinita o meno oppure se contiene uno 0…). Vediamo il seguente esempio:

<?php

 $val2 = empty($val1); /\*$val1 non è ancora definita, quindi $val2 sarà vero\*/

 echo $val2;

 ?>

Un’altra funzione importante è **is\_int()** la quale restituisce un booleano (vero se il parametro che gli viene passato è un intero, falso altrimenti). Analogamente si ha per la verifica per un reale, una stringa, un array, un campo numerico e così via. Vediamo in merito i seguente esempio:

<?php

 /\*controllo se campo numerico\*/

 $x=is\_numeric(3);

 echo $x;

 /\*controllo se campo real\*/

 $y=is\_float(3.3);

 echo $y;

 /\*controllo se campo stringa\*/

 $z=is\_string("ss");

 echo $z;

 /\*controllo se campo intero\*/

 $w=is\_int("a");

 echo $w;

 /\*controllo se campo array\*/

 $a=is\_array(1);

 echo $a;

 ?>

Infine è anche possibile ottenere al volo il tipo posseduto da una variabile, come viene mostrato di seguito, utilizzando la funzione **gettype()**:

<?php

 $x=3;

 echo gettype($x);

 ?>

Anche PHP5, come altri tipi di linguaggi, è in grado di lavorare con le stringhe. E’ risaputo che una stringa è una sequenza di caratteri. Un esempi di stringa è la seguente:

 “luce”

PHP5 mette a disposizione del programmatore alcune funzioni per la manipolazione delle stringhe. Vediamo una breve carellata delle più utilizzate. Se vogliamo per esempio visualizzare la lunghezza di una stringa utilizziamo la funzione **strlen** come viene mostrato di seguito:

<?php

 $str="luce";

 echo strlen($str);

 ?>

Verrà visualizzato a video il numero 4 (4 caratteri della stringa). Supponiamo ora di voler estrarre una sottostringa dalla stringa principale. Per esempio, supponiamo di avere a disposizione la seguente stringa:

 str=”lucedeimieiocchi”;

e di voler estrarre la sottostringa “luce”. Per effettuare tale operazione si utilizza la funzione **substr** passandogli come argomento la stringa originale, da quale carattere iniziare l’estrapolazione, e quanti caratteri si vogliono estrarre.

<?php

 $str="lucedeimieiocchi";

 $str2=substr($str,0,4);

 echo $str2;

 ?>

E’ importante notare che i caratteri vanno contati a partire da zero. Se si desidera invece sostituire parti di stringa si utilizza la funzione **str\_replace** come viene mostrato di seguito:

<?php

 $str="lucedeimieiocchi";

 $str2=str\_replace("luce","buio",$str);

 echo $str2;

 ?>

Come si nota facilmente, la sottostringa “luce” verrà sostituita con la sottostringa “buio”. Se ora desidero ottenere la posizione in cui la sottostringa “miei” risiede nella stringa originaria è sufficiente utilizzare la funzione **strpos()** come viene mostrato di seguito:

<?php

 $str="lucedeimieiocchi";

 $str2=str\_replace("luce","buio",$str);

 $pos=strpos($str, "miei");

 echo $pos;

 ?>

Supponiamo ora di avere a disposizione la seguente stringa:

 str=”ciao\_mondo”

vogliamo spezzettare la stringa in due stringhe diverse “ciao” e “mondo” in base al carattere speciale **underscore (\_)**. Tale operazione in informatica viene denominata **splittaggio**. Solitamente esiste una funzione per permettere lo splittaggio di una stringa. Tale funzione è la funzione **split()**. Anche in PHP era presente tale funzione che però è stata deprecata in PHP5 e sostituita con la funzione **explode** come mostra il seguente esempio:

<?php

 $str="ciao\_mondo";

 $vett= explode('\_', $str);

 echo $vett[0];

 echo $vett[1];

 ?>

Come mostra l’esempio, il risultato è un vettore di stringhe. Esistono poi altre funzioni per lavorare con le stringhe. Per maggiori informazioni si rimanda il lettore ai vari manuali più approfoniti disponibili su Internet. Ad ogni modo alcune funzioni molto utilizzate, oltre a quelle appena citate, sono:

* **trim()** che permette di eliminare gli spazi bianchi nella stringa.
* **Ltrim()** che permette di eliminare gli spazi bianchi all’inizio della stringa.
* **Rtim()** che permette di eliminare gli spazi bianchi alla fine della stringa.

**Strtolower()** che permette di mettere inminuscoo tutti i caratteri della stringa.

* **Strtoupper()** che permette di mettere in maiuscolo tutti i caratteri della stringa.

Analogamente per quanto visto con le stringhe, esisotno delle funzioni per lavorare sugli array. Vediamo alcune funzioni interessanti tramite degli esempi:

<?php

 $vett = array('x', 'y', 'z');

 echo count($vett);

 ?>

Questo semplicissimo esempio mostra l’utilizzo della funzione **count** per contare il numero di elementi presenti nell’array.

<?php

 $vett=array('a', 'b', 'c');

 $arr1 = array\_reverse($vett);

 for($i=0; $i<count($vett);$i++)

 echo $arr1[$i];

 ?>

La funzione **array\_reverse** permette di invertire l’ordine degli elementi del vettore. Per esempio se si ha il seguente vettore:

 $x=[‘a’, ‘b’, ‘c’];

con tale funzione si ottiene un nuovo vettore del seguente tipo:

 $y=[‘c’, ‘b’, ‘a’];

Per poter ordinare un vettore si utilizzainvece la funzione **sort** come viene mostrato di seguito:

<?php

 $vett=array(3,2,1);

 $vett1=sort($vett);

 for($i=0; $i<count($vett1);$i++)

 echo $vett1[$i];

 ?>

Per ricercare un elemento particolare in un vettore si utilizza la funzione **in\_array()**, come viene mostrato di seguito:

<?php

 $vett=array(3,2,1);

 $trovato=in\_array(1, $vett);

 echo $trovato;

 ?>

Si presti attenzione che la funzione restituisce 1 (true) in caso che ha trovato l’elemento, altrimenti restituisce 0 (false). Se si desidera invece fondere i valori presenti in un array su una singola stringa bisogna effettuare l’operazione opposta dello splitting ossia bisogna usare la funzione opposta dell’explode. Tale funzione è l’**implode** e viene utilizzata nel seguente modo:

<?php

 $vett=array(3,2,1);

 $str=implode(' ', $vett);

 echo $str;

 ?>

Come si può facilmente notare, il carattere “spazzietto” (‘ ‘) viene utilizzato per separare le varie sottostringhe (gli elementi dell’array) della stringa stessa. Altre funzioni utili sono:

* **Array\_pop**, che permette di estrarre l’ultimo elemento dell’array.
* **Array\_push**, che permette di accodare in fondo nuovi elementi nell’array.
* **Array\_merge**, che permette di fondere gli elementi di due array.
* **Array\_shift**, che permette di estrarre il primo elemento dell’array e quindi di shiftware tutti gli altri.

Vediamo ora di terminare questa breve carrellata sulle principali funzioni offerte da PHP con la gestione delle date. E’ bene ricordare che la gestione delle date in PHP avviene tramite l’utilizzo del classico **timestamp** di Unix. Il timestamp è un numero intero che rappresenta il numero di secondi trascorsi dal primo gennaio 1970. La funzione che permette di estrarre il timestamp attuale è la funzione **time**, che viene mostrata di seguito:

<?php

 $time=time();

 echo $time;

 ?>

Dato il timestamp, per ottenere la data odierna è sufficiente utilizzare la funzione **date** :

<?php

 $time=time();

 $data = date('d M y - H:i', $time);

 echo $data;

 ?>

Talefunzione ha alcuni parametri illustrati qui di seguito:

**Y** anno su 4 cifre

**y** anno su 2 cifre

**N** mese numerico (1-12)

 **m** mese numerico su 2 cifre (01-12)

**F** mese testuale ('January' - 'December')

 **M** mese testuale su 3 lettere ('Jan' - 'Dec')

**d** giorno del mese su due cifre (01-31)

**J** giorno del mese (1-31)

**W** giorno della settimana, numerico (0=dom, 6=sab)

**l** giorno della settimana, testuale ('Sunday' - 'Saturday' )

 **D** giorno della settimana su 3 lettere ('Sun' - 'Sat')

**H** ora su due cifre (00-23) **G**ora (0-23)

**i** minuti su due cifre (00-59)

**s** secondi su due cifre (00-59)

**Per effettuare una verifica sulla data inserita è opportuno utilizzare la funzione checkdate.**

**Un altro costrutto di base è l’istruzione require(). Tale istruzione include e valuta il file specificato. Un’altra istruzione identica è include, con la sola differenza che require blocca l’esecuzione della pagina se il file specificato è mancante mentre include non lo permette. Vediamo un esempio:**

<?php

 require 'file\_require.php';

 echo $x;

 ?>

**ed il file\_require.php contiene il seguente frammento di codice:**

**<?php**

 **$x=34;**

 **?>**

**I costrutti e le istruzioni fin qui enunciate sono quelle di base del linguaggio. Chiaramente, ci sono in commercio guide più dettagliate a cui fare tranquillamente riferimento. Vediamo ora di descrivere brevemente il concetto di funzione. Consideriamo il seguente esempio:**

function fun ($arg\_1, $arg\_2, ..., $arg\_n)

{

echo "Esempio di funzione...\n";

return $retval;

}

Il precedente esempio mostra una generica funzione denominate fun che ha come parametri di ingresso ‘n’ argomenti (‘n’ variabili). Una funzione è un qualcosa che ha un tot di ingressi, li elabora, e produce determinate uscite. Questo modo di programmare (programmazione procedurale) tipica dei moderni linguaggi di programmazione permette di rendere il codice snello e leggibile in quanto permette di suddividere le operazioni svolte dallo script tra più funzioni richiamate opportunamente all’interno della porzione di codice principale del programma. Per default le variabili vengono passate alle funzioni per **valore**. Questo comporta che, una volta terminata l’esecuzione della funzione, il valore modificato dalla funzione della variabile stessa si perde. Passando invece la variabile per **riferimento** è possibile non perdere tale valore. Il passaggio della variabile per riferimento avviene ante cedendo al nome della variabile il carattere speciale ‘&’, come viene mostrato di seguito:

Function fun(&$string)

{

$string .= ’xxx.’;

}

$str = ’stringa, ’;

fun($str);

echo $str; // outputs ’stringa, xxx.’

**Giunti a questo punto possiamo fare un balzo in avanti ed iniziare ad entrare nel vivo della programmazione orientata agli oggetti visto che PHP è un linguaggio object oriented. Iniziamo con il definire il costrutto di base presente nella programmazione orientata agli oggetti: la classe. Una classe è una collezione di variabili e metodi per poter operare su tali variabili. Un oggetto è un’istanza della classe. Per poter lavorare con una classe bisogna quindi prima istanziarla e successivamente si può operare usando i metodi sulle variabili con la usuale notazione puntata. Vediamo il seguente esempio:**

<?php

class test

{

function f1()

{

echo "Test";

}

}

$obj = new test;

$obj->f1();

?>

**Come mostra l’esempio, abbiamo creato una classe di nome test al cui interno vi è una funzione (metodo) denominate f1. Tale funzione stampa a video la stringa Test. Successivamente, viene creato un oggetto ($obj) che viene usato per eseguire la funzione f1. Se nella classe test vi fosse stata presente una variabile denominata per esempio $var\_test, allora tramite l’oggetto $obj si poteva accedere al contenuto di tale variabile nel seguente modo:**

 **$obj.$var\_test**

**Spesso, soprattutto per chi ha lavorato assiduamente nel campo della programmazione orientata agli oggetti, si è costretti a costruire classi derivate da altre classi. L’ereditarietà è una caratteristica importante della programmazione orientata agli oggetti. Per esempio, è possibile creare una classe generica denominata veicolo che contiene tutte le caratteristiche di un generico veicolo, e successivamente creare una classe automobile che deriva naturalmente della classe veicolo (visto che l’automobile è un tipo particolare di veicolo) e che contiene le proprietà generiche di un’automobile. In PHP l’erediterietà si ottiene tramite la parola chiave extends. Si veda in merito il seguente esempio:**

**<?php**

 **class auto extends veicolo**

 **{**

 **var $motore;**

 **var $targa;**

 **var $colore;**

 **}**

 **?>**

**In questo esempio la classe auto deriva dalla classe veicolo. Chiusa la sezione sulle classi analizziamo ora la parte più importante di PHP. La gestione dei database e l’interazione con i files HTML. Partiamo da quest’ultima.**

**Un form in HTML è sostanzialmente un insieme di elementi di input (textbox, checkbox, radiobox,…). In poche parole in form permette all’utente di inserire determinate informazioni. Pensate, a titolo esemplificativo, alle schermate di registrazione per l’acquisto on line di un determinato prodotto. Una volta inseriti i dati e dopo aver cliccato sul pulsante “Submit” vengono inviati ad una applicazione presente su un server (solitamente uno script) che li elabora e restituisce un risultato. L’URL (indirizzo) di tale script viene specificato nell’attributo action del tag HTML form. Per maggiori chiarimenti invito il lettore a leggersi il manuale HTML scaricabile gratuitamente sul mio stesso sito. Si veda in merito il seguente esempio:**

<form action="elabora.php" method="get">

<input type="text" name="campione">

<input type="submit" name="bInvia" value="Invia i dati">

</form>

**Lo script PHP si chiama elabora.php ed i dati vengono inviati utilizzando il metodo GET. Per la trasmissione dei dati verso lo script esistono fondamentalmente due metodi:**

1. **Il metodo post**
2. **Il metodo get**

**Il metodo POST permette di inviare i dati della form nello standard input dell’applicazione destinataria e pertanto non sono visibile nell’indirizzo URL. Cosa molto importante: non ci sono limiti sulla quantità di dati da inviare. Con il metodo GET i dati vengono accodati all’indirizzo e pertanto sono nettamente visibile nello stesso. Per questo motivo la quantità di dati inviabili con il metodo GET è limitato ai 256 caratteri dell’URL stesso. Per l’interprete PHP l’utilizzo di un metodo piuttosto che l’altro non fa alcuna differenza. Le interpreta e le memorizza in variabili globali all’interno dello script.**

**I seguenti esempi mostrano, lato PHP, la gestione delle variabile con il metodo GET e con il metodo POST.**

**LATO HTML:**

<form action="lettura.php" method="post">

<input type="text" name="nome">

<input type="checkbox" name="nuovo" value="si">

<input type="submit" name="submit" value="invia">

</form>

**LATO PHP:**

**<?php**

 **Var $test = $\_POST[‘nome’];**

 **echo $test;**

**?>**

**$\_POST deve essere visto come un array.**

**Un altro concetto importante in PHP sono le sessioni. E’ risaputo che il protocollo universalmente utilizzato per la navigazione su internet è il protocollo HTTP. Tale protocollo è però state less ossia privo di stato. Questo vuol dire che quando l’utente abbandona la pagina web le transazioni che si sono verificate vengono perdute. Per risolvere il problema sono stati inventati i cookie, ossia dei normali file di testo in cui salvare le informazioni di sessione. Il gruppo di lavoro di PHP, per risolvere tale problema e semplificare la questione senza scomodare troppo i cookie, ha introdotto un’intelligente gestione delle sessioni. In poche parole, quando viene chiamata la funzione start\_session oppure session\_register i dati di sessione vengono registrati nella memoria di sessione. Il percorso dove risiede la memoria di sessione viene specificato nel file php.ini nella direttiva session.save\_path. Quando una sessione viene avviata gli viene assegnato un ID di sessione. Pertanto il supporto delle sessioni PHP consiste nel mantenere certi dati attraverso accessi successivi. Tale concetto è molto importante anche per velocizzare gli accessi alle pagine. Al visitatore che accede ad una nostra pagina verrà assegnato un id univoco ossia l’id di sessione.**

**Dalla versione PHP 4.1.0 la variabile $\_SESSION è disponibile come variabile globale proprio come $\_GET e $\_POST.** Perché la variabile $\_SESSION sia utilizzabile bisogna modificare il file php.ini nella directory WINDOWS settando session.auto\_start =1 e creando, se non esiste, una directory temp dove vengono registrate le variabili di sessione. Vediamo il seguente esempio chiarificatore:

<?php

session\_start();

if (!session\_is\_registered('contatore'))

 {

 session\_register('contatore');

 $\_SESSION[contatore] = 1;

}

else {

 $\_SESSION[contatore]++;

}

?>

<html>

 <head>

 <title>Sessioni</title>

 </head>

 <body>

 <?php

 echo "<h2>Pagina visitata $\_SESSION[contatore] volte.</h2>

 <A HREF=\"session\_cookie.php\">clicca qui</A>";

 ?>

 </body>

</html>

**L’altro concetto importante è il concetto di database. La possibilità di interagire con i database è una delle caratteristiche più interessanti di PHP. I database relazionali sono lo strumento per definizione utilizzato per conservare i dati di qualsiasi dimensione. Il PHP ci fornisce la possibilità di connetterci con un numero elevatissimo di database (MySQL, Oracle, ….) Per prima cosa è necessario collegarsi al database in questione. Questa operazione in PHP è semplicissima e viene mostrata di seguito:**

 **mysql\_connect(server, utente, password);**

La connessione si realizza attraverso la funzione **mysql\_connect**. Che necessita dei seguenti parametri:

* + **server** : l’URL del database
	+ **utente** : nome di accesso
	+ **password** : password dell’utente

**Una volta stabilita la connessione è possibile effettuare la selezione del database su cui intendiamo lavorare.** Per questo si usa la funzione **mysql\_select\_db** che necessita dei seguenti parametri:

* + **nomedb**: il nome del db al quale vogliamo connetterci
	+ **connessione**: l'identificativo di connessione (cioè quello che abbiamo ottenuto da mysql\_connect)

La sintassi è la seguente:

 mysql\_select\_db(nomedb,connessione);

Questa funzione **restituisce** un valore booleano in funzione dell’esito dell’operazione stessa. Giunti a questo punto è possibile effettuare delle query (interrogazioni) sul db selezionato. Tale operazione è possibile svolgerla tramite la funzione mysql\_query i cui parametri sono:

* + **query**:query da eseguire
	+ **connessione**: identificativo di connessione

Anche questa funzione **restituisce** un valore, per il quale però dobbiamo distinguere due possibilità rispetto al tipo di query che abbiamo lanciato:

* + **query di interrogazione** (SELECT, SHOW, EXPLAIN, DESCRIBE), la funzione restituisce un **identificativo del risultato** (cioè un'altra variabile di tipo resource), che ci servirà successivamente, se la query è andata a buon fine; se invece MySql ha rilevato degli errori, la funzione restituisce FALSE
	+ **query di aggiornamento** (INSERT, UPDATE, DELETE), la funzione restituirà in ogni caso un valore booleano, ad indicare se l'esecuzione è andata a buon fine oppure no

A questo punto appare evidente che è necessario anche verificare la corretta esecuzionedella query. Se vogliamo sapere quante righe sono state restituite da un’operazione di lettura possiamo, per esempio, usare la funzione **mysql\_num\_rows()**. Se invece abbiamo eseguito una query di aggiornamento (INSERT, UPDATE, DELETE) e vogliamo sapere quante righe sono state modificate, possiamo usare **mysql\_affected\_rows(connessione)**, che ci restituisce il numero di righe modificate dall'ultima query di aggiornamento.

Una volta concluse le varie operazioni su db è possibile chiudere la connessione allo stesso. Tale operazione viene svolta tramite la funzione **mysql\_close()** che ha come unico parametro l’id della connessione. Attenzione però: questa funzione viene pochissimo usata in quanto PHP si preoccupa da solo di chiudere la connessione al db una volta terminato l’esecuzione dello script. Il seguente script mostra un esempio completo di accesso ad un database in PHP.

<?php

 $dbUtente="root";

 $passwd="";

 $dbNome="test";

 $dbHost="localhost";

 //tentativo connessione a db Test

 if(!($conn=mysql\_connect($dbHost, $dbUtente, $passwd)))

 {

 echo "Error connessione!!";

 }

 //seleziono db corrente

 if(!mysql\_select\_db($dbNome, $conn))

 {

 echo "Error db!!";

 }

 //esecuzione query

 $query="select \* from Test";

 $result=mysql\_query($query, $conn);

 echo $result;

 //chiusura connessione

 mysql\_close($conn);

 ?>