PHP

1. Introduzione

PHP è un acronimo ricorsivo che significa **P**HP: **H**ypertext **P**reprocessor. Quindi il PHP è un preprocessore di ipertesti, nel senso che l'elaborazione del codice PHP produce in output un codice HTML puro. O almeno questo è lo scopo per cui questo linguaggio è nato e la sua funzione principale.

PHP è un linguaggio di programmazione web lato server, è un linguaggio di scripting con un buon interprete e una forte predisposizione all'interfacciamento con i database (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.).

L'interprete e il linguaggio PHP sono software e specifiche aperti e liberi di essere scaricati e utilizzati, oltreché cross-platform (è possibile ad esempio sviluppare su windows una applicazione web in php destinata alla produzione su macchine UNIX/Linux).

Sintassi Base del PHP

Il codice PHP viene eseguito sul server e restituisce in output stringhe html pure da essere inviate al browser.

Un blocco di script PHP inizia sempre con **<?php** e finisce sempre con **?>** e può essere posto ovunque all'interno del documento.

Un file .php può inoltre contenere semplice codice html fuori dai blocchi degli script che rimarranno invariati nell'interpretazione del codice.

Vediamo un primo esempio di codice con un file html con scritto solo il classico “Hello, World!”

```
<html>
<body>

<?php
echo "Hello World";
?>

</body>
</html>
```

Ogni linea in PHP termina con un punto e virgola, che funziona anche da separatore di comandi. I file PHP devono avere estensione .php, altrimenti l'interprete non li riconoscerà come tali e il codice PHP rimarrà ignorato.

I commenti lungo il codice PHP seguono la stessa sintassi del C/C++

```
<?php
//commento di linea

/*
Questo è un blocco di
commenti (lungo quanto
vi pare!!!
*/
?>
```

I comandi di base per scrivere output di testo in PHP sono due: **echo** e **print**. Il comando echo è lo stesso utilizzato nel nostro primo esempio.

Variabili

Le variabili si utilizzano per memorizzare informazioni, come stringhe numeri o insiemi di queste. Quando se ne dichiara una può essere utilizzata per tutta la lunghezza di uno script.

È molto semplice individuare una variabile in una porzione di codice PHP, perché tutti i nomi di variabile devono iniziare con $. Il modo corretto di dichiarare una variabile è dunque:

```
$variabile = valore;
```

Vediamo adesso un esempio di codice in cui vengono dichiarate una variabile contenente un numero ed una contenente una stringa:

```
<?php
$txt = "Hello World!";
$x = 16;
?>
```

Il PHP è uno dei linguaggi comunemente definiti ***Loosely Typed***. Questo significa che non c'è bisogno di dichiarare una variabile prima di utilizzarla e che il PHP converte automaticamente la variabile al tipo corretto di dato, a seconda del valore inserito.

In un linguaggio ***Strongely Typed***, come il C ad esempio, è necessario dichiarare il nome e il tipo di una variabile prima di poterla utilizzare.

Regole per i nomi delle variabili

- devono iniziare con una lettera dell'alfabeto o con underscore
- possono contenere solo numeri, lettere e underscore
- non possono contenere spazi

Variabili Stringa

Le variabili stringa sono utilizzate per memorizzare e manipolare il testo. Vediamo innanzitutto un semplice esempio di dichiarazione e utilizzo:

```
<?php
$txt = "Hello World";
echo $txt;
?>
```

Questo semplice script dichiara la variabile $txt, gli assegna il valore “Hello World” e poi visualizza il contenuto della variabile.

L'***operatore di concatenazione*** (.) viene utilizzato per mettere insieme più valori.

```
<?php
$txt1 = "Hello World!";
$txt2 = "What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

L'esempio illustrato dichiara due variabili e poi le visualizza concatenate con in mezzo uno spazio “ “.

Questo è dunque un operatore semplicissimo da usare e spesso utile per costruire le frasi da visualizzare in output

La ***funzione strlen()*** calcola il numero di caratteri di una stringa. Il seguente esempio di codice restituisce 12, il numero di caratteri di cui è composta la stringa “Hello world!”.

```
<?php
echo strlen("Hello world!");
?>
```

Questa funzione viene molto utilizzata quando in un ciclo si vuole scorrere una stringa carattere per carattere, operando su ognuno di questi.

La ***funzione strpos()*** è utilizzata per cercare caratteri o stringhe all'interno di una stringa. Se la ricerca ha esito positivo, la funzione restituisce la posizione numerica della prima occorrenza trovata, altrimenti restituisce FALSE.

```
<?php
echo strpos("Hello world!","world");
?>
```

Questo esempio di codice visualizza il numero 6, che è la posizione del primo carattere (partendo da zero) della prima occorrenza della stringa “world”.

Operatori PHP

Gli operatori di un linguaggio vengono utilizzati per lavorare sui valori. Vediamo una lista completa degli operatori disponibili in PHP:

Operatori Aritmetici

Operatore	Descrizione	Esempio	Risultato
+	Addizione	x=2 x+2	4
-	Sottrazione	x=2 5-x	3
*	Moltiplicazione	x=4 x*5	20
/	Divisione	15/5 5/2	3 2.5
%	Modulo (resto divisione intera)	5%2 10%8 10%2	1 2 0
++	Incremento	x=5 x++	x vale 6
--	Decremento	x=5 x--	x vale 4

Operatori di Assegnazione

Operatore	Esempio	È come fare
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Operatori di confronto

Operatore	Descrizione	Esempio
==	È uguale a	5==8 restituisce false
!=	È diverso da	5!=8 restituisce true
>	È maggiore di	5>8 restituisce false
<	È minore di	5<8 restituisce true
>=	È maggiore o uguale a	5>=8 restituisce false
<=	È minore o uguale a	5<=8 restituisce true

Operatori Logici

Operatore	Descrizione	Esempio
&&	and	x=6 y=3 (x < 10 && y > 1) restituisce true
||	or	x=6 y=3 (x==5 || y==5) restituisce false
!	not	x=6 y=3 !(x==y) restituisce true

2. Istruzioni condizionali

Le istruzioni condizionali sono utilizzate per eseguire azioni diverse a seconda delle condizioni che si verificano. Il PHP supporta le seguenti strutture condizionali:

- **Struttura if:** utilizzata per eseguire codice solo se si verifica una condizione.
- **Struttura if...else:** utilizzata per eseguire determinate istruzioni se una condizione si verifica e altre altrimenti.
- **Struttura if...elseif....else:** questo blocco supporta la scelta fra varie opzioni.
- **Struttura switch:** analogo al precedente, ma (forse) con una migliore organizzazione dei blocchi di codice

La struttura *if*

La struttura if viene utilizzata per eseguire una porzione di codice solo se una specificata condizione risulta vera.

```
if (condizione)
{
     codice da eseguire
}
```

Vediamo un esempio in cui se il giorno corrente è Lunedì, viene visualizzata una scritta di buon inizio settimana

```
$d = date("D");
if ($d=="Mon")
{
     echo "Buon inizio settimana!”;
}
```

La struttura *if...else*

La struttura if...else si utilizza per decidere quale codice eseguire a seconda che una condizione specificata sia vera oppure falsa.

```
if(condizione)
{
     codice per condizione vera
}
else
{
     codice per condizione falsa
}
```

il seguente esempio dice se un numero è pari oppure dispari, calcolando il resto della divisione per 2:

```
$d = 3;
if($d % 2 == 0)
{
     echo "pari";
}
else
{
     echo "dispari";
}
```

La struttura if...elseif...else

Questa struttura permette di gestire diversi blocchi di codice scegliendone uno da eseguire in base ai valori delle condizioni specificate.

```
if (condizione 1)
{
     codice condizione 1 vera
}
elseif (condizione 2)
{
     codice condizione 2 vera
}
else
{
     codice condizione 2 falsa
}
```

L'esempio seguente augura buon weekend agli utenti che lo eseguono di venerdì, “Buona domenica” a quelli che lo eseguono di domenica, “Buona Giornata” in tutti gli altri momenti

```
$d = date("D");
if($d=="Fri")
{
  echo "Buon weekend!";
}
elseif ($d=="Sun")
{
  echo "Buona Domenica!";
}
else
{
  echo "Buona giornata!";
}
```

La struttura switch

La struttura switch viene utilizzata per eseguire diverse porzioni di codice a seconda del valore di una determinata variabile in esame.

```
switch($variabile)
{
case $valore1:
     blocco codice 1;
     break;
case $valore2:
     blocco codice 2;
     break;
…
default:
     blocco codice default;
}
```

La struttura switch controlla il valore della variabile in esame. Se $variabile == $valore1, allora viene eseguita la prima porzione di codice e poi si esce dal blocco switch, effetto ottenuto tramite l'esecuzione dell'istruzione ***break***.

Se $variabile == $valore2, allora viene eseguita la seconda porzione di codice e poi si esce dal blocco switch. Così via per tutti i valori presenti. Se $variabile non è uguale a nessuno dei valori specificati, si esegue il blocco default.

Vediamo un esempio in cui si utilizza lo switch:

```
switch ($n)
{
case 1:
  echo "Numero 1";
  break;
case 2:
  echo "Numero 2";
  break;
case 3:
  echo "Numero 3";
  break;
default:
  echo "Il numero non è fra 1 e 3”;
}
```

Esercizi

Creare una pagina HTML contenente una serie di link ad altrettante pagine PHP, ognuna avente come output uno degli esercizi qui proposti:

1. Dato un numero verificare se è pari o dispari
2. Scambiare due valori fra loro (ordinare due valori)
3. Determinare il maggiore fra tre numeri.
4. Ordinare tre valori

3. Array

Un array è un tipo speciale di dato che può memorizzare un numero predefinito di valori in una variabile sola.

Quando si ha a che fare con una lista di oggetti (nomi, numeri, indirizzi, etc...) l'array è la variabile da usare:

```
$name1 = “Andrea”;
$name2 = “Silvia”;
$name3 = “Alessia”;
```

In questo esempio si utilizzano 3 variabili diverse per memorizzare 3 nomi. Ma se i nomi fossero stati 300? Se dovessimo fare una ricerca tra tutti i nomi per trovare ad esempio quelli che iniziano per “S”?

La soluzione in tutti questi casi è quella di utilizzare un array.

Con quello infatti, tutte le variabili sono accessibile tramite il solo nome dell'array e sono differenziate solo dall'indice, che indica la loro posizione all'interno dell'array.

```
$names = array(“Andrea”, “Silvia”, “Alessia”, “Gianmarco” );
```

In PHP esistono tre tipi di array:

- **Numerici**: sono array con indice numerico, ove quindi gli elementi sono numerati.
- **Associativi**: sono array in cui l'indice è un ID di un qualunque tipo
- **Multidimensionali**: Sono array contenenti uno o più array

Array Numerici

Un array è numerico quando i suoi elementi si distinguono tramite un indice numerico. Ci sono due modi per crearne uno:

1. con una dichiarazione implicita:

   ```
   $names = array(“Andrea”, “Silvia”, “Alessia”, “Gianmarco” );
   ```

   Nel cui caso la numerazione è progressiva e parte da zero.

2. con una dichiarazione esplicita:

   ```
   $names[0] = “Andrea”;
   $names[1] = “Silvia”;
   $names[2] = “Alessia”;
   $names[3] = “Gianmarco”;
   ```

   In cui si specifica ogni valore per ogni posizione che si vuole occupare

Dichiarato un array, si possono utilizzare i valori ivi contenuti semplicemente riferendosi al suo indice:

```
$names = array(“Andrea”, “Silvia”, “Alessia”, “Gianmarco” );
echo $names[0] . “ è il mio nome”;
```

Array Associativi

In un array associativo, ogni valore è associato ad un ID unico, che può essere un qualsiasi tipo. Con gli array associativi, si possono utilizzare i valori come chiavi e assegnarli dei valori, come nell'esempio seguente:

```
$eta = array(“Alessia” => 7, “Gianmarco”=> 3)
```

Così è possibile utilizzare i nomi come chiavi e le età come valori dell'array, che potranno essere riutilizzati in questo modo:

```
$nome = “Alessia”;
echo $nome.”ha .”$eta[$nome].” anni”;
```

Questo codice visualizza come output la scritta

```
Alessia ha 7 anni
```

Array Multidimensionali

Negli array multidimensionali, ogni elemento dell'array può essere anche un array esso stesso. Inoltre anche ogni elemento del sotto array può essere un array e così via.

```
$famiglieFamose = array(
“Fantozzi”=>array(“Ugo”, “Pina”, “Mariangela”),
“Flintstones”=>array(“Fred”, “Wilma”, “Ciotolina”),
“Addams”=>array(“Gomez”, ”Morticia”, ”Mercoledì”, ”Pugsley”)
);
```

L'array famiglieFamose contiene 3 array che contengono ognuna alcune stringhe. Per accedere al nome del primo elemento della famiglia Flintstones dovremo semplicemente scrivere:

```
echo $famiglieFamose[ 'Flinstones' ][0];
```

4. Istruzioni Iterative

Le iterazioni sono esecuzioni consecutive di un blocco di codice per un numero (possibilmente) finito di volte.

In PHP si hanno i seguenti costrutti sintattici per implementare delle interazioni:

- **while -** ripete un blocco di codice finché una (pre)condizione specificata è vera.
- **do...while -** ripete un blocco di codice finché una (post)condizione specificata è vera.
- **for** - ripete un blocco di codice per un numero specificato di volte.
- **foreach** - ripete un blocco di codice per ogni elemento di un array

Il ciclo while

Il ciclo while controlla prima una condizione specificata e poi inizia a ripetere un blocco di codice, finché la condizione risulta vera.

Questo significa che se la condizione è inizialmente falsa il blocco di codice non viene eseguito neanche una volta!!

```
while(condizione)
{
     blocco di codice
}
```

Vediamo un esempio di utilizzo di questo costrutto molto comune:

```
$i=1;
while($i<=5)
{
  echo "Numero " . $i . "<br />";
  $i++;
}
```

che visualizza 5 scritte con i numeri che vanno da 1 a 5.

Il ciclo do...while

Il ciclo do...while esegue prima una iterazione del blocco di codice e poi controlla una condizione specificata, iniziando a ripetere il blocco, finché la condizione risulta vera.

Questo significa che anche se la condizione è inizialmente falsa il blocco di codice viene eseguito almeno una volta!!

```
do{
    blocco di codice
}
while(condizione);
```

Vediamo un esempio di utilizzo di questo costrutto, che è solitamente poco utilizzato se non in casi particolari in cui la condizione controlla dei valori che vengono assegnati durante l'esecuzione del blocco.

```
$numero=0;
do{
    $numero+=3;
}
while($numero<10);
echo $numero;
```

che cosa visualizzerà questo codice come output?

Il ciclo for

Il ciclo for si utilizza quando è noto a priori il numero di iterazioni che devono essere eseguite su un blocco di codice.

```
for( init, condizione, incremento)
{
     blocco di codice
}
```

Di solito il ciclo for è molto apprezzato per la sua sintassi compatta e per la chiarezza della sua condizione. Vediamo un esempio in cui ripeteremo una frase per 10 volte:

```
for($i=1; $i<=10;$i++)
{
     echo “ciao<br />”;
}
```

Il ciclo foreach

Il ciclo foreach viene utilizzato per eseguire un blocco di codice per tutti gli elementi di un array:

```
foreach( $array as $value)
{
  blocco di codice con $value;
}
```

Per ogni iterazione del ciclo, la variabile $value assume uno alla volta tutti i valori di ogni elemento dell'array.

```
$colori = array(“rosso”, “giallo”, “verde”, “blu” );
foreach ($colori as $c)
{
  echo “colore “.$c.” <br />”;
}
```

Esercizi

Creare una pagina HTML contenente una serie di link ad altrettante pagine PHP, ognuna avente come output uno degli esercizi qui proposti:

1. Il minimo e massimo di un assegnato vettore di numeri interi,
2. La somma e la media aritmetica di un assegnato vettore di numeri interi,
3. Un vettore di numeri interi nell'ordine inverso rispetto a quello in cui è assegnato,
4. Un vettore di numeri interi seguiti ciascuno dagli aggettivi "grande" e "piccolo" a seconda che tali numeri siano rispettivamente maggiori, o minori, di 10.
5. Una tabella con sei colonne e venti righe, con ciascuna delle 120 caselle contenente la dicitura "riga x, colonna y" con x e y opportunamente sostituiti dalla posizione della casella nella tabella,
6. una tabella contenente una matrice assegnata di interi,
7. una tabella contenente la matrice trasposta (ottenuta, cioè scambiando righe e colonne) di una matrice assegnata di interi

5. Gestione dei Form con PHP

Le variabili PHP $_GET e $_POST sono utilizzate per reperire le informazioni inserite nei form HTML.

Queste due variabili predefinite sono infatti due array associativi che contengono automaticamente ogni informazione inviata da un form HTML rispettivamente tramite metodo GET e POST.

Vediamo un banale esempio di form e gestione dei dati tramite PHP:

```
<form action="welcome.php" method="post">
Nome: <input type="text" name="name" />
Cognome: <input type="text" name="surname" />
<input type="submit" />
</form>
```

le informazioni saranno inviate al click sul submit alla pagina welcome.php, che deve essere presente nella stessa directory ove questo codice viene implementato:

```
Benvenuto, <?php echo $_POST["name"].” “.$_POST["surname"]; ?>.
```

La validazione dei form, come ad esempio il controllo dei campi vuoti o dei campi mail o data, dovrebbe essere effettuata tramite scripts lato client (ad esempio tramite javascript).

Questo per incrementare la velocità di elaborazione e ridurre il carico dei server.

La variabile $_GET si usa in maniera analoga per reperire i valori inviati da un form tramite method=”get”.

La differenza fra i metodi di invio GET e POST è che mentre i valori in una connessione POST sono inviati nascosti, quelli in una connessione GET sono appiccicati all'URL di invio (specificato nell'attributo action del form).

Così in una situazione analoga alla precedente, ma con metodo GET specificato:

```
<form action="http://thissite.com/welcome.php" method="get">
Nome: <input type="text" name="name" />
Cognome: <input type="text" name="surname" />
<input type="submit" />
</form>
```

Le informazioni scritte nel form sono inviate tramite il seguente URL:

```
http://thissite.com/welcome.php?name=pinco&surname=pallino
```

Ovviamente il metodo GET non è adatto per l'invio di dati strettamente riservati o password. E non è inoltre in grado di gestire trasferimenti contenenti più di 100 bytes di informazioni.

Esercizi

1. Form per l'inserimento dei dati personali e relativa visualizzazione dei dati inviati
2. Creazione di una pagina web per la ricerca di un valore su Google
3. Autenticazione (nome utente e password) su una pagina web con controllo dei dati

6. Funzioni PHP

La reale forza del PHP si scopre studiando le 700 e più funzioni predefinite di cui dispone e che gli permettono di intervenire con semplicità in praticamente tutti gli aspetti di interazione con la rete, con i dati e i database e con la macchina su cui questo codice viene eseguito.

Come prima introduzione all'argomento si vedrà come è possibile in PHP creare una propria funzione per eseguire un compito prestabilito, così da familiarizzare con la sintassi e essere subito pronti per una migliore organizzazione del codice.

La sintassi di definizione delle funzioni PHP è la seguente:

```
funzione NomeFunzione( $parametri )
{
  codice da eseguire;
}
```

E' importante utilizzare dei nomi per le funzioni che riflettano il suo reale comportamento, in modo da orientarsi meglio fra tutte quelle che si finirà solitamente ad usare scrivendo codice PHP.

L'unica regola da ricordare è che come tutti gli identificatori, il nome di una funzione deve cominciare con una lettera o un underscore, **non** con un numero!

Per eseguire il codice di una funzione, bisognerà utilizzare una chiamata alla funzione stessa, effettuabile tramite il suo nome, seguito da parentesi, più eventuali parametri:

```
NomeFunzione();
```

Vediamo subito un esempio di una funzione molto semplice:

```
<?php
function scriviNome()
{
  echo "Andrea Diamantini";
}

echo "Il mio nome e' ";
scriviNome();
?>
```

Vediamo adesso un esempio in cui si passano alcuni parametri ad una funzione:

```
function scriviDati($nome, $cognome)
{
  echo “Mi chiamo”.$nome.” “.$cognome;
}

scriviDati(“Pinco”, “Pallino”);
$n=”Paolo”;
$c=”Rossi”;
scriviDati($n,$c);
```

Per permettere alle funzioni di ritornare un valore è necessario utilizzare l'istruzione return:

```
function somma($a,$b)
{
  $sum=$a+$b;
  return $sum;
}
$primo = 4;
$secondo = 7;
echo “La somma di “.$primo.” e “.$secondo.” porta “.somma($primo,
$secondo);
```

Funzioni per data e ora

La funzione PHP date() viene utilizzata per formattare un orario e/o una data, ricreando un opportuno timestamp, cioè una opportuna sequenza di caratteri utilizzata per denotarne il formato (es: gg/mm/aa).

La sintassi della funzione date() è semplicissima:

```
date(format, timestamp)
```

Parametro	Descrizione
format	Obbligatorio. Specifica il formato del timestamp.
timestamp	Opzionale. Specifica un timestamp, con default data e ora corrente.

Il parametro ***format*** specifica come visualizzare data/ora. Vediamo tutti i parametri che può prendere:

d	Due cifre per il giorno del mese (da 01 a 31)
D	Tre lettere per rappresentare un giorno (Mon, Tue, Wed, Thu, Fri, Sat, Sun)
j	Il giorno del mese, senza zeri eccedenti (da 1 a 31)
l ('L' minuscola)	Rappresentazione testuale del giorno
N	La rappresentazione ISO 8601 del giorno (1 = lunedì, 2 = martedì, etc..)
S	Il suffisso ordinale Inglese adatto al giorno (2 caratteri, st, nd, rd, th)
w	Rappresentazione numerica del giorno (0 = domenica, 1 = lunedì, etc...)
z	Il giorno dell'anno (da 0 a 365)
W	Il numero della settimana dell'anno secondo ISO 8601
F	Rappresentazione testuale del mese
m	Rappresentazione numerica del mese (da 01 a 12)
M	Rappresentazione testuale breve del mese (3 lettere)
n	Rappresentazione numerica del mese (da 1 a 12)
t	Il numero del giorno per il mese
L	Anno bisestile (1 sì, 0 no)
o	Il numero dell'anno secondo lo standard ISO 8601
Y	Rappresentazione dell'anno con 4 cifre

y	Rappresentazione dell'anno con 2 cifre
a	am o pm minuscoli
A	AM o PM maiuscoli
B	Swatch Internet time (da 000 a 999)
g	Ora del giorno (da 1 a 12)
G	Ora del giorno (da 0 a 23)
h	Ora del giorno (da 01 a 12)
H	Ora del giorno (da 00 a 23)
i	Minuti (da 00 a 59)
s	Secondi (da 00 a 59)
e	timezone identifier (UTC, Atlantic/Azores, etc..)
I (i maiuscola)	Ora legale (1 = ora legale, 0 = ora solare)
O	Differenza in ore dall'ora di Greenwich (GMT)
T	Timezone della macchina che esegue il PHP
Z	Differenza in secondi dall'ora di Greenwich (GMT)
c	Data ISO 8601 (esempio: 2004-02-12T15:19:21+00:00)
r	Data RFC 2822 (esempio: Thu, 21 Dec 2000 16:01:07 +0200)
U	Secondi trascorsi dalla Unix Epoch (January 1 1970 00:00:00 GMT)

Il parametro opzionale timestamp serve a specificarne uno invece di usare quello per data e ora corrente.

Il comando UNIX mktime() restituisce il timestamp per una data, corrispondente al numero di secondi tra la “UNIX Epoch” (January 1 1970 00:00:00 GMT) e il tempo specificato.

```
mktime(ore,minuti,secondi,mesi,giorni,anni, bisestile)
```

Vediamo dunque alcuni esempi di utilizzo della funzione date() e della funzione mktime().

```
<?php
$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
echo "Domani è ".date("Y/m/d", $tomorrow);
?>
```

7. Tecniche avanzate con PHP

In questa sezione troviamo alcune tecniche di gestione avanzata del codice e delle risorse tramite PHP. Ognuna delle sezioni presentate è indipendente e può quindi servire come approfondimento personale oltreché per imparare una nuova tecnica con PHP.

Server Side Includes (SSI)

Il contenuto di un file PHP può essere inserito e utilizzato dentro un altro file PHP prima che il server lo esegua, tramite le funzioni ***include()*** o ***require()***.

Le due funzioni sono identiche in tutto e per tutto tranne per come gestiscono gli errori:

- include() genera uno warning, ma lo script continua l'esecuzione.
- require() genera un errore fatale, interrompendo l'esecuzione dello script

Queste due funzioni sono utilizzate per creare funzioni, header e elementi da riutilizzare in più pagine.

I cosiddetti Server Side Includes risparmiano un sacco di lavoro, facilitando un approccio moderno e sistemico nell'organizzazione del codice delle proprie applicazioni lato server.

Esempio

funzioni.php

```
<?php
function Nome($n)
{
  echo $n;
}
?>
```

prova1.php

```
<?php
```

```
include(“funzioni.php”);
Nome(“Andrea”);
echo “Benvenuto!”;
?>
```

prova2.php

```
<?php
require(“funzioni.php”);
Nome(“Andrea”);
echo “Benvenuto!”;
?>
```

Questi script presuppongono l'esistenza del file funzioni.php nella stessa directory ove essi si trovano. Nel caso di un errore, ad esempio la mancata presenza del file, il primo restituisce il seguente output:

```
Warning: include(funzioni.php) [function.include]:
failed to open stream:
No such file or directory in /home/website/prova1.php on line 2

Warning: include() [function.include]:
Failed opening 'funzioni.php' for inclusion (include_path='.;C:\php5\pear')
in /home/website/prova1.php on line 2

Benvenuto!
```

Quindi eseguendo comunque lo script come il “Benvenuto” finale, esecuzione dell'ultimo echo dimostra.

Il secondo file con direttiva require in caso di errore si comporta invece:

```
Warning: require(funzioni.php) [function.require]:
failed to open stream:
No such file or directory in /home/website/prova2.php on line 2

Fatal error: require() [function.require]:
Failed opening required 'funzioni.php' (include_path='.;C:\php5\pear')
in /home/website/prova2.php on line 2
```

Fermando dunque l'esecuzione dello script in cui un errore si verifica.

Gestione File

The funzione ***fopen()*** viene utilizzata per aprire i file in PHP. Il primo parametro di questa funzione contiene il nome del file da aprire, mentre il secondo specifica in che modo il file dovrà essere aperto:

```
<?php
$file = fopen( "welcome.txt" , "r" );
?>
```

Un file può essere aperto nei seguenti modi:

Modo	Descrizione
r	Sola lettura. Parte dall'inizio del file.
r+	lettura/scrittura. Parte dall'inizio del file.
w	Sola scrittura. Apre e cancella il contenuto del file o ne crea uno nuovo vuoto se non esiste.
w+	lettura/scrittura. Apre e cancella il contenuto del file o ne crea uno nuovo vuoto se non esiste.
a	Append. Apre e scrive alla fine del file o crea un nuovo file se non esiste.
a+	Lettura/Append. Legge dall'inizio e scrive dal fondo, conservando il contenuto del file.
x	Sola scrittura. Crea un nuovo file o restituisce FALSE se il file esiste già.
x+	lettura/scrittura. Crea un nuovo file o restituisce FALSE se il file esiste già.

Se la funzione fopen() non è in grado di aprire il file specificato, restituisce 0 (false), informazione che può essere utilizzata come nel seguente esempio:

```
$file = fopen("welcome.txt", "r") or exit("Impossibile aprire il
file!");
```

La funzione ***fclose()*** viene utilizzata per chiudere un file:

```
$file = fopen("test.txt","r");

// qualcosa da fare con questo file
```

```
fclose($file);
```

La funzione ***feof()*** controlla che non si sia raggiunti la fine del file (eof = end of file). Restituisce un valore booleano di conferma.

```
if (feof($file))
     echo "Raggiunta fine del file";
```

Per leggere una singola linea dal un file di testo, è necessario utilizzare la funzione ***fgets()***. Dopo una chiamata a questa funzione il file pointer si muove alla linea successiva, rendendo possibile costruire un ciclo di lettura con questa funzione:

```
$file = fopen("welcome.txt", "r") or exit("Unable to open file!");

while(!feof($file))
{
  echo fgets($file). "<br />";
}

fclose($file);
```

La funzione ***fgetc()*** legge un singolo carattere da un file, spostando automaticamente il file pointer all'inizio del prossimo carattere.

```
$file = fopen("welcome.txt","r") or exit("Unable to open file!");

while (!feof($file))
{
  echo fgetc($file);
}

fclose($file);
```

File upload

Con PHP è possibile ad esempio eseguire l'upload di un file dalla macchina client alla macchina server. Questa operazione viene eseguita creando due semplici script: il form HTML per la scelta del file da parte dell'utente e il codice di upload del file dal client al server.

Il form HTML che permette l'upload di un file può essere definito così:

```
<form action="upload.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" />
<br />
<input type="submit" name="submit" value="Submit" />
</form>
```

Da notare le seguenti particolarità del codice HTML sopra:

- L'attributo enctype dell'elemento form specifica quale contenuto verrà spedito dal form. Il valore “multipart/form-data” viene utilizzato per i dati binari generici.
- Il valore “file” dell'attributo type dell'elemento input permette la selezione di un elemento dal file system.

In generale, permettere agli utenti un upload di file sul server è un grosso rischio e bisognerebbe permetterlo solo ad utenti autenticati e a certe condizioni.

Lo script “upload.php” chiamato dal form al submit conterrà il codice (appunto) per l'upload.

```
if ($_FILES["file"]["error"] > 0)
{
  echo "Error: " . $_FILES["file"]["error"] . "<br />";
}
else
{
  echo "Upload: " . $_FILES["file"]["name"] . "<br />";
  echo "Type: " . $_FILES["file"]["type"] . "<br />";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
}
```

Utilizzando l'array globale PHP $_FILES si può fare l'upload di file dal client a un server remoto. Il primo parametro è il nome dell'input form e il secondo indice può essere uno fra “name”, “type”, “size”, “tmp_name”, “error”.

- $_FILES["file"]["name"] – nome del file di cui fare l'upload
- $_FILES["file"]["type"] – tipo del file
- $_FILES["file"]["size"] – dimensione in byte del file
- $_FILES["file"]["tmp_name"] – nome della copia temporanea del file sul server
- $_FILES["file"]["error"] – il codice di errore risultato dall'upload

Questo esempio è comunque banale e per motivi di sicurezza bisognerebbe comunque implementare controlli sul tipo, la dimensione e sul numero di file uploadabili da ogni utente prima di permettere una tale libertà.

Il codice precedente crea una copia temporanea del file sul server che viene distrutta al termine dello script PHP. Per mantenerla in memoria occorre copiarla in un'altra directory

```
if (file_exists("upload/" . $_FILES["file"]["name"]))
{
  echo $_FILES["file"]["name"] . " already exists. ";
}
else
{
  move_uploaded_file($_FILES["file"]["tmp_name"],
  "upload/" . $_FILES["file"]["name"]);
  echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
}
```

Gestione Cookies

Un cookie (letteralmente, biscottino) è un piccolo file che il server inserisce nel computer di un utente per identificarlo alla prossima visita. Ogni volta che un browser richiede una pagina, il server risponde inviando anche un cookie insieme alla pagina. Tramite il PHP è possibile sia creare che ricevere dei cookie.

Come è possibile creare un cookie? E' possibile creare un cookie semplicemente tramite la funzione setCookie(). Questa funzione per essere efficace deve essere eseguita PRIMA del tag <html> della pagina a cui il cookie si riferisce.

Vediamone la sintassi:

```
setcookie(name, value, expire, path, domain);
```

Vediamo qualche esempio di utilizzo dei cookie per cominciarne a capire l'utilità. In questo primo esempio viene creato un cookie che contiene il mio nome e che sarà valido per un'ora

```
<?php
setcookie("nome", "Andrea", time()+3600);
?>

<html>
.....
```

Il valore del cookie è automaticamente URLencoded quando questo viene inviato. Se si vuole prevenire questa funzione automatica, occorre usare la funzione setrawcookie().

Nel prossimo esempio si inserisce un cookie con un dato che abbia validità per un mese (1 mese = 60 ss * 60 mm * 24 hh * 30 gg)

```
$expire = time()+60*60*24*30;
setcookie("nome", "Andrea", $expire);
```

Per andare a controllare i valori impostati precedentemente tramite cookie si utilizza l'array associativo $_COOKIE, che contiene come indici i nomi delle variabili impostate tramite cookie e al suo interno i loro valori:

```
echo $_COOKIE["nome"];
```

```
// oppure, per vedere tutti i cookies
print_r($_COOKIE);
```

Un'altra cosa solitamente utile da controllare sui cookie che si trovano già sulla macchina client è se hanno un certo valore già impostato:

```
if (isset($_COOKIE["nome"]))
  echo "Ciao " . $_COOKIE["user"] . "!<br />";
else
  echo "Benvenuto per la prima volta qui!<br />";
```

Se infine si vuole cancellare un cookie (tramite PHP ovviamente, non tramite le impostazioni e le utility di un browser), bisogna fargli credere di essere scaduto. Infatti, i cookie che superano il loro “expire time” si auto cancellano automaticamente:

L'esempio imposta il valore di scadenza del cookie precedente con il nome memorizzato ad un'ora fa:

```
setcookie("nome", "", time()-3600);
```

Gestione Sessioni

Una variabile di sessione PHP viene utilizzata per immagazzinare le informazioni necessarie ad una sessione utente. Le variabili di sessione contengono informazioni su un singolo utente e sono disponibili in tutte le pagine di una applicazione.

Quando si lavora davanti al computer, magari scrivendo un testo con un semplice editor, il sistema operativo ha informazioni precise su chi sei e cosa puoi fare, sa quando è stata avviata l'applicazione che stai usando e si accorgerà di quando la chiuderai.

Tutte queste informazioni sono sconosciute su Internet (per fortuna) per la natura stessa delle connessioni HTTP che vengono usate (*stateless*). Le sessioni PHP risolvono questo problema permettendo di immagazzinare informazioni come il nome utente, le pagine visitate, gli acquisti selezionati o altro per un uso successivo.

In ogni caso, queste informazioni sono temporanee e saranno cancellate quando l'utente lascerà il sito. Per salvare informazioni in maniera permanente occorrerà utilizzare un file o meglio ancora un database.

Le sessioni funzionano creando un ID unico (UID) per ogni visitatore e memorizzando le variabili relative a questo, basandosi sul suo UID. Questo viene inoltre memorizzato in un cookie temporaneo o inviato tramite l'URL.

Prima di poter immagazzinare informazioni in una sessione PHP è necessario iniziarne una:

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

<html>
...
```

Anche qui, l'istruzione session_start() deve essere eseguita PRIMA del tag <html>. Questa funzione registra la sessione utente nel server, assegnando un UID per quella sessione.

Per inserire informazioni di sessione, è necessario utilizzare la variabile $_SESSION, l'ennesimo array associativo che incontriamo in PHP:

```
$_SESSION['visite']=1;
```

Nell'esempio seguente si crea un contatore di visite come può essere la variabile qui sopra e poi lo si aggiorna ad ogni visita di quella pagina, controllando con la funzione isset() se è stata già impostata (pagina già visitata, incremento del contatore) oppure no (contatore visite a 1).

```
<?php
session_start();
if(isset($_SESSION['visite']))
    $_SESSION['visite']++;
else
    $_SESSION['visite']=1;
?>
```

Per cancellare alcuni dati di sessione è possibile utilizzare la funzione unset():

```
unset($_SESSION['visite']);
```

Per terminare completamente una sessione, distruggendo quindi tutti i dati relativi, si utilizza la funzione session_destroy().

```
session_destroy();
```

Invio Mail

Il linguaggio PHP permette di inviare email direttamente da uno script, tramite la funzione mail(). Ovviamente per rendere questo possibile, è necessario che vi sia installato un mail server funzionante.

```
mail(to, subject, message, headers, parameters);
```

Parametro	Descrizione
to	Obbligatorio. Specifica il destinatario della mail.
subject	Obbligatorio. Specifica l'oggetto della mail.
message	Obbligatorio. Specifica il messaggio da inviare. Ogni linea deve essere separata da un LF (\n) e non deve superare i 70 caratteri.
headers	Opzionale. Specifica parametri aggiuntivi come From, CC, BCC. Devono essere separati da un carattere CRLF (\r\n)
parameters	Opzionale. Specifica eventuali parametri addizionali di invio per il programma di posta.

Il modo più semplice di inviare una mail con PHP è quello di scrivere una mail di testo. Nell'esempio sottostante, prima vengono dichiarate le variabili che si utilizzeranno ($to, $subject, $message, $from, $headers) e poi vengono utilizzate nella funzione mail().

```
<?php
$to = "te@esempio.com";
$subject = "mail di prova";
$message = "Ciao! Se ti arriva questa, siamo i draghi del PHP!”;
$from = "me@esempio.com";
$headers = "From: $from";
mail($to,$subject,$message,$headers);
echo "Posta inviata.";
?>
```

Questa semplice tecnica di invio delle mail può essere implementata per creare ad esempio un form di invio mail in un sito. Basterà prendere i valori inseriti da un utente in alcuni campi specifici e assegnarli a delle variabili come nell'esempio sopra.

Ovviamente questo semplicissimo metodo può comportare un sacco di inconvenienti, quindi occore in ogni caso validare i contenuti dei campi immessi dagli utenti prima di avviare il comando di invio mail.

Gestione degli Errori

Il meccanismo standard di gestione degli errori in PHP è molto semplice: ad ogni errore nel codice viene inviato un messaggio al browser con nome del file, numero di linea e descrizione dell'errore avvenuto.

Quando si creano script e applicazioni web, la gestione degli errori è una parte importante. Se un'applicazione manca del controllo degli errori sembra solitamente poco professionale e può inoltre aprire il varco a possibili problemi di sicurezza.

I metodi più utilizzati di controllo degli errori in PHP sono

- l'utilizzo della funzione die()
- l'utilizzo dei trigger
- la visualizzazione degli errori

Se si prova ad esempio ad aprire con opzione “r” un file inesistente

```
$file=fopen("prova.txt","r");
```

il PHP riporta il seguente errore

```
Warning: fopen(prova.txt) [function.fopen]: failed to open stream:
No such file or directory in fileXX on line YY
```

Lo stesso non avviene se il programmatore controlla l'esistenza del file prima di aprirlo

```
if(!file_exists("prova.txt"))
  die("File not found");
else
  $file=fopen("prova.txt","r");
```

Che invece di un messaggio di errore del codice riporta la scritta “File not found”, decisa in fase di creazione dell'applicazione.

Ovviamente, interrompere lo script e lasciare un semplice messaggio di errore non è sempre la strada migliore e a volte non è neanche possibile, per non lasciare l'applicazione in uno stato inconsistente.

Il PHP fornisce come alternativa delle funzioni di gestione degli errori, i cosiddetti “*error handler*”.

Queste funzioni, eseguite quando occorre un particolare errore, devono essere in grado di gestire

almeno due parametri:

1. error level
2. error description

ma possono in generale arrivare a gestirne fino a cinque, includendo anche:

3. file
4. line number
5. error context

Vediamo la sintassi:

```
err_funct(err_level, err_message, err_file, err_line, err_context);
```

Parametro	Descrizione
err_level	Valore numerico obbligatorio. Specifica il tipo di errore, come descritto nella tabella seguente.
err_message	Obbligatorio. Specifica il messaggio di errore per l'utente.
err_file	Opzionale. Nome del file dove è successo l'errore.
err_line	Opzionale. Numero di linea del file dove è successo l'errore.
err_context	Opzionale. Specifica un array contenente le variabili e i loro valori definiti quando l'errore è avvenuto.

Error Levels

Valore	Costante	Descrizione
2	E_WARNING	Errore di run-time non falale. L'esecuzione dello script non sarà terminata.
8	E_NOTICE	Informazione a run-time. Lo script ha trovato qualcosa che potrebbe essere un errore, ma che potrebbe anche permettere la normale esecuzione dello script.
256	E_USER_ERROR	Errore fatale, generato dall'utente tramite esecuzione della funzione trigger_error()
512	E_USER_WARNING	Errore **non** fatale, generato dall'utente tramite esecuzione della funzione trigger_error()
1024	E_USER_NOTICE	Informazione non fatale, generata dall'utente tramite esecuzione della funzione trigger_error()
4096	E_RECOVERABLE_ERROR	Errore che può essere raccolto da un handler definito dall'utente.
8191	E_ALL	Tutti gli errori e gli warning tranne quelli di livello E_STRICT (che diventeranno parte di E_ALL nel PHP 6.0)

Adesso, creiamo una funzione per gestire gli errori:

```
function customError($errno, $errstr)
{
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Ending Script";
  die();
}
```

Questa funzione semplicemente prende error level e error description, li scrive su schermo e termina lo script. Per lanciare questa funzione ad ogni occorrenza di un errore, bisogna installare un error handler per quella funzione, tramite la funzione set_error_handler().

```
set_error_handler("customError");
```

In questo modo la funzione customError gestirà tutti i messaggi di errore. Se invece volessimo assegnarla alla gestione di soli di errori di tipo E_WARNING, avremmo dovuto scrivere

```
set_error_handler("customError", E_WARNING);
```

Negli script che hanno a che fare con l'input utente è molto comodo definire un errore e lanciarlo quando avviene un input sbagliato. In PHP questo può essere realizzato tramite la funzione trigger_errno().

```
$test=2;
if ($test>1)
  trigger_error("Value must be 1 or below");
```

E ancora, aggiungendo un secondo parametro alla funzione può essere specificato il tipo di errore da eseguire. Possibili tipi per gli errori utente sono:

- E_USER_ERROR – Errore fatale generato dall'utente a runtime. Provoca l'interruzione dell'esecuzione dello script.
- E_USER_WARNING - Errore non fatale generato dall'utente a runtime. Non interrompe l'esecuzione dello script.
- E_USER_NOTICE - Default. Informazione di runtime generata da un input utente.

In questo esempio, se la variabile test è più grande di uno, viene lanciato un errore E_USER_WARNING per cui è stato installato un gestore apposito che interrompe lo script (anche se questo non sarebbe il comportamento PHP predefinito).

```
//error handler function
function customError($errno, $errstr)
{
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Ending Script";
  die();
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
```

```
  trigger_error("Value must be 1 or below",E_USER_WARNING);
```

In generale il PHP scrive un un file di log (il cosiddetto error_log, la cui locazione è impostata nel file php.ini) tutte le occorrenze di errori che avvengono.

E' possibile in ogni caso, tramite la funzione error_log() inviare gli errori ad un altro file, oppure ad una destinazione remota (un email, ad esempio).

Vediamo come realizzare quest'ultimo esempio:

```
//error handler function
function customError($errno, $errstr)
{
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "me@esempio.com","From: php@esempio.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
  trigger_error("Value must be 1 or below",E_USER_WARNING);
```

8. PHP e DATABASE

Il linguaggio PHP è notoriamente in grado di assicurare una semplice interazione con i database. Ma, se questo discorso è vero in generale, vale ancora di più per il database server MySQL, con cui si sposa in maniera semplice, veloce ed efficasissima.

La loro “intesa di coppia” è talmente buona e testata, che da anni dominano insieme la scena del Web, come linguaggio di scripting e database più diffuso della Rete.

Per interagire con un database MySQL, bisogna prima di tutto creare una connessione, tramite la funzione mysql_connect().

mysql_connect(servername , username , password);

Parametro	Descrizione
servername	Opzionale. Spedifica il server a cui connettersi (default ="localhost:3306")
username	Opzionale. Specifica l'username da utilizzare nella connessione. Valore di default è l'username che detiene il processo server.
password	Opzionale. Specifica la password di connessione dell'utente. Vuota di default.

Vediamo subito un esempio di connessione, il cui risultato viene memorizzato in una variabile e in cui la seconda parte dello script provoca una interruzione immediata dello stesso qualora la connessione non andasse a buon fine.

```
<?php
$con = mysql_connect("localhost","andrea","abc123");
if (!$con)
{
  die('Impossibile Connettersi: ' . mysql_error());
}

...
?>
```

La connessione verrà chiusa automaticamente alla fine dello script. Per chiuderla prima, sarà necessario utilizzare la funzione mysql_close();

```
mysql_close($con);
```

Quando la connessione con il database server è attiva, è possibile interagire con questo tramite il linguaggio SQL e la funzione PHP mysql_query() che esegue sul DB server il codice SQL inserito come argomento.

Ad esempio sarà possibile, se l'utente con cui siamo connessi ne ha diritto, creare un database con questo semplice comando:

```
<?php
$con = mysql_connect("localhost","andrea","abc123");
if (!$con)
{
  die('Connessione impossibile: ' . mysql_error());
}

if ( mysql_query( "CREATE DATABASE mioDb" , $con) )
{
  echo "Creato Database mioDb";
}
else
{
  echo "Errore nella creazione del database: " . mysql_error();
}

...
mysql_close($con);
?>
```

Un database può poi contenere una o più tabelle, che possono essere create in maniera del tutto analoga con l'istruzione mysql_query().

Per assicurarsi però di creare le tabelle sul database appena creato, occorrerà selezionarlo con l'istruzione mysql_select_db().

```
// Create table
mysql_select_db("mioDb", $con);
$sql = "CREATE TABLE Persona
(
personID int  NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
Nome     varchar(15),
Cognome  varchar(15),
Anni     int
)";

// Execute query
mysql_query( $sql , $con );
```

Per inserire nuovi record dentro una tabella si procede con l'istruzione SQL INSERT .

```
...
mysql_query("INSERT INTO Persona (Nome, Cognome, Anni)
VALUES ('Peter', 'Griffin', '35')");

mysql_query("INSERT INTO Persona (Nome, Cognome, Anni)
VALUES ('Glenn', 'Quagmire', '33')");
...
```

Ecco un semplice esempio di come un Form HTML può essere utilizzato per inserire dati dentro un Database:

Il Form HTML

```
<html>
<body>

<form action="insert.php" method="post">
nome: <input type="text" name="nome" />
cognome: <input type="text" name="cognome" />
<input type="submit" />
</form>

</body>
</html>
```

Il file insert.php

```
<?php
$con = mysql_connect("localhost","andrea","abc123");
if( !$con )
{
  die('Impossibile connettersi: ' . mysql_error());
}
mysql_select_db( "mioDb" , $con );

$sql="INSERT INTO Persone (Nome, Cognome)
VALUES
('$_POST[nome]','$_POST[cognome]')";

if( !mysql_query( $sql , $con ) )
{
  die('Errore: ' . mysql_error());
}
echo "Aggiunto 1 record";

mysql_close( $con );
?>
```

Il comando SQL SELECT viene utilizzato per selezionare dati da un database. Come già detto, il comando per eseguire comandi SQL su database MySQL è l'istruzione mysql_query(), con la differenza in questo caso specifico che il comando restituirà il risultato della query che sarà da memorizzare in una nuova variabile.

Da questo risultato sarà possibile estrarre una riga per volta tramite il comando mysql_fetch_array(). Ognuna delle righe della tabella così ottenute sarà un array associativo con indici dei campi i nomi delle colonne corrispettive nella tabella SQL. Sarà dunque semplice estrarre i dati per il trattamento e la visualizzazione.

Nell'esempio seguente vediamo un esempio di esecuzione di query sul database e successiva visualizzazione dei dati.

```
<?php
$con = mysql_connect("localhost","andrea","abc123");
if (!$con)
{
  die('Impossibile connettersi: ' . mysql_error());
}

mysql_select_db("mioDb", $con);

$risultato = mysql_query("SELECT * FROM Persone");

while( $row = mysql_fetch_array($risultato) )
{
  echo $row['nome'] . " " . $row['cognome'];
  echo "<br />";
}

mysql_close($con);
?>
```

Database e ODBC

ODBC è l'acronimo di Open Data Base Connectivity ed è un API per la connessione ad una sorgente dati. Tramite ODBC è possibile connettersi a qualunque tipo di fonte per i dati della propria applicazione, da un database nel server MS SQL, ad un database MySQL, fino ad un database MS Access e perfino ad un foglio elettronico.

Per creare una connessione ODBC è necessario utilizzare i tools che il proprio sistema operativo offre, come quelli presenti nel Pannello di Controllo dei sistemi Windows o agli ODBC tools presenti negli altri sistemi operativi.

La connessione PHP ad una risorsa ODBC si effettua tramite la funzione odbc_connect(), che prende 4 parametri in ingresso.

```
odbc_connect( source_name , username , password , cursor_type );
```

Analogamente, la funzione odbc_exec() viene utilizzata per eseguire una istruzione SQL su una connessione ODBC attiva.

```
$conn = odbc_connect( 'northwind' , '' , '' );
$sql = "SELECT * FROM customers";
$rs = odbc_exec( $conn , $sql );
```

La variabile PHP $rs conterrà il risultato della query, che sarà una tabella creata con i dati risultanti dall'esecuzione del codice SQL. Per selezionare una riga da suddetta tabella, dovremo utilizzare il comando odbc_fetch_row().

```
odbc_fetch_row( $rs );
```

Analogamente, per ottenere i campi dalla riga selezionata, dovremo utilizzare il comando odbc_result(), che lavora nei modi seguenti:

```
$primo = odbc_result( $rs , 1 );
```

assegna alla variabile di output il valore presente nel primo campo della riga selezionata.

```
$name = odbc_result( $rs , "Name" );
```

assegna alla variabile di output il valore presente nel campo di nome “Name” della riga selezionata.

Quando alla fine si vuole chiudere una connessione con la risorsa ODBC, è sufficiente utilizzare il comando odbc_close():

```
odbc_close( $conn );
```

Vediamo infine un esempio di come creare una connessione ad una risorsa ODBC di nome ProvaODBC, ottenere dati dalla sua tabella “tabClienti” e visualizzarli in una tabella HTML.

```
<html>
<body>

<?php
$conn = odbc_connect('ProvaODBC','','');
if ( !$conn )
{
    exit("Connessione NON riuscita: " . $conn);
}
```

```
$sql = "SELECT * FROM tabClienti";
$rs = odbc_exec( $conn , $sql );
if( !$rs )
{
    exit("Errore nel codice SQL");
}
echo "<table><tr>";
echo "<th>Nome</th>";
echo "<th>Cognome</th></tr>";
while ( odbc_fetch_row( $rs ) )
{
    $nome = odbc_result( $rs , "nome" );
    $cognome = odbc_result( $rs ,"cognome" );
    echo "<tr><td>$nome</td>";
    echo "<td>$cognome</td></tr>";
}
odbc_close( $conn );
echo "</table>";
?>

</body>
</html>
```