

Capitolo 1 29/01/2009

*IPSLA “Antonio Pacinotti” - Pistoia*

# Corso sul linguaggio PHP

*Gestione Web di un Magazzino*



III Area

**Classe V** - Gestione Web di un  
magazzino di materiale elettronico

Gualtiero Lapini

# Corso sul linguaggio PHP

*Progettazione e collaudo*

**Scopo dell'esercitazione:** Simulare un caso “*reale*” di progettazione di pagine web “*dinamiche*” per la gestione di un magazzino di materiale elettronico.

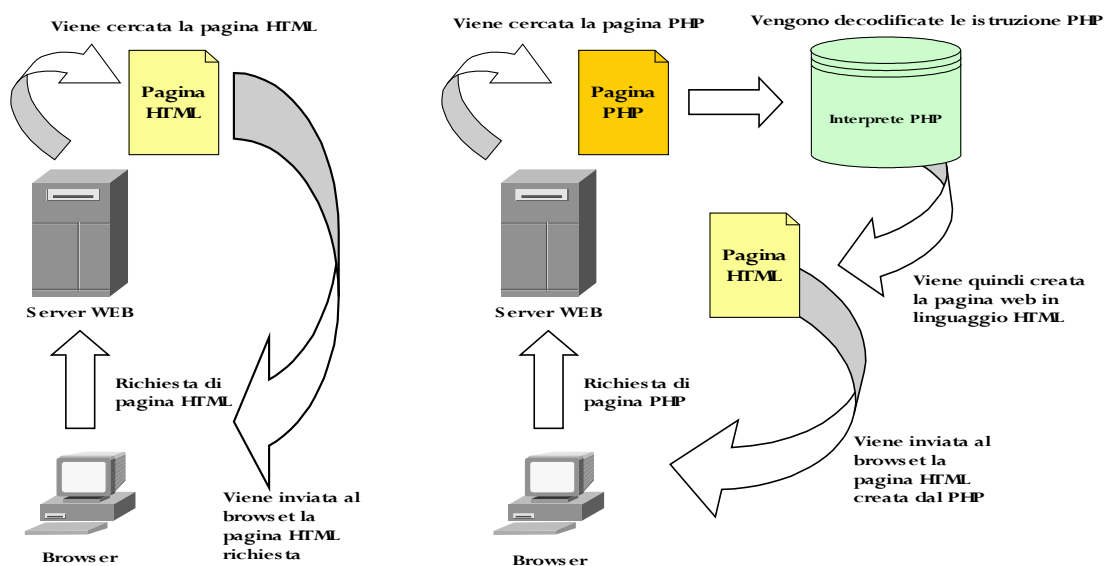
**Procedimento:** Partendo dal precedente corso riguardante il linguaggio MySQL utilizzato per la gestione di un database, vengono apprese le basi del linguaggio PHP per la creazione di pagine web “*dinamiche*”.

Lasciamo da parte per ora le nozioni riguardanti il linguaggio MySQL per la gestione dei database, che peraltro riprenderemo verso la parte finale di questo corso.

Partiamo dalle basi del linguaggio PHP che attualmente è uno dei più diffusi per la creazione di pagine web dinamiche. Abbiamo scelto questo linguaggio per la grande facilità di interfacciarsi con un database (nel nostro caso MySQL).

A differenza del linguaggio HTML con cui si possono creare delle pagine web ma di tipo **statico**, ovvero la pagina verrà sempre visualizzata così come è stata creata dal programmatore, le pagine scritte con un linguaggio come il PHP possono diventare **dinamiche**, cioè presentarsi ogni volta in modo diverso a seconda delle situazioni che si sono create in quel momento: scelta da parte dell'utente che naviga, data e ora in cui esso si connette, pulsanti che vengono premuti od altri oggetti che sono stati modificati dall'utente e, soprattutto, in base ai dati presenti in quel momento nel database.

Vediamo un grafico che schematizza cosa succede in un server web nel caso venga richiesta dal browser una pagina in linguaggio HTML oppure in linguaggio PHP:



**Pagina HTML**

**Pagina PHP**

Come schematizzato in figura si vede che, se il browser richiede una pagina web scritta in HTML al server remoto, esso la cerca e la invia così com'è al browser che utilizzerà le istruzioni HTML per ricostruire la pagina e visualizzarla sullo schermo del computer. Mentre se il browser richiede una pagina scritta in PHP al server remoto, esso la cerca e, una volta localizzata, la passa al preprocessore PHP, il quale in pratica traduce le istruzioni PHP, da esse crea una nuova pagina web, in linguaggio HTML, che verrà poi inviata al browser che l'ha richiesta. Il browser quindi riceve una pagina scritta in HTML e non la nostra pagina sorgente in PHP.

Iniziamo a vedere come si scrivono le pagine in PHP.

Per prima cosa occorre ricordarsi che, mentre per l'HTML occorre dare ai files l'estensione **\*.htm** oppure **\*.html**, per i files in PHP occorre obbligatoriamente assegnare l'estensione **\*.php**, altrimenti non verranno riconosciuti dall'interprete o preprocessore PHP.

Ma cosa scriviamo dentro a questi files? Semplicemente le istruzioni HTML come già conosciamo, però oltre a queste possiamo aggiungere delle istruzioni in linguaggio PHP, inserendole all'interno di uno specifico tag. Questo tag è contrassegnato all'inizio da **<?php** ed al termine da **?>**. Il preprocessore leggerà e interpreterà solo le istruzioni inserite tra questi due tag e le tradurrà quindi in istruzioni HTML. Le pagine PHP quindi possono essere anche interpretate direttamente da un browser, con la differenza che ciò che compare fra questi due tag verrà ignorato. La stessa cosa succede se si caricano queste pagine in un server web in cui non è presente il servizio PHP.

Vediamo quindi il listato del nostro primo programma in linguaggio PHP.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Esercitazioni PHP</title>
    <meta name="description" content="Corso PHP">
    <meta name="keywords" content="">
    <meta name="author" content="Gualtiero Lapini">
    <meta name="generator" content="Crimson Editor">
  </head>
  <body>
    <p align="center">
      <em>
        <font face="Tunga"size="7">
          Corso III Area
        </font>
      </em>
    </p>
    <DIV style="TEXT-ALIGN: center">
      <span style="FONT-WEIGHT: bold">
        GESTIONE MAGAZZINO
      </span>
      <br>
      <span style="FONT-STYLE: italic">
        Linguaggio PHP
      </span>
    </DIV>
    <DIV style="TEXT-ALIGN: center">
      <SPAN style="FONT-STYLE: italic">
        </SPAN><br>&nbsp;
      </DIV>
    <span style="COLOR: rgb(255,0,0)" >
      Il testo che compare tra le due linee sottostanti è
      relativo ad istruzioni PHP
    </span>
    <br>
    =====
    <br>
  </body>
</html>

```

```

<?php
    echo "Questo è il primo programma scritto in linguaggio PHP<br>";
?>
=====
<br>
<span style="COLOR: rgb(204,0,0)" >
    Questo invece è di nuovo testo in HTML.
</span>
<br>
<br>
</body>
</html>
    
```

La parte di istruzioni in linguaggio PHP è evidenziata in colore rosso ed è racchiusa tra due righe in colore azzurro. Nei prossimi esercizi modificheremo solo la parte di testo racchiuso tra queste due righe, per cui verranno indicate solo le istruzioni in PHP, tralasciando tutto il resto della pagina web che, peraltro, è necessaria per la corretta visualizzazione della stessa.

Cosa succede se si salva la pagina nel server web (indirizzo del server 172.16.10.200, però siccome lo stiamo provando in locale sulla stessa macchina server allora l'indirizzo può essere direttamente il classico localhost) e da un qualsiasi client web con browser Internet Explorer, Firefox, Mozilla, Opera, ecc. se ne chiede la visualizzazione?

Ecco il risultato: la parte scritta in nero tra le due linee è quella proveniente dal linguaggio PHP. Se andiamo a visualizzare, tramite menù il sorgente HTML della pagina vedremo che...



```

....testo eliminato....
<span style="COLOR: rgb(255,0,0)" >
Il testo che compare tra le due linee
sottostanti è relativo ad istruzioni PHP
</span><br>
=====<br>
Questo è il primo programma scritto in linguaggio PHP<br>
=====
<span style="COLOR: rgb(204,0,0)" >
Questo invece è di nuovo testo in HTML.
</span>
<br>
....altro testo eliminato....
    
```

La parte scritta in PHP è completamente scomparsa! Come mai? Semplice, l'interprete PHP l'ha decodificata e trasformata in puro HTML, quindi il client non riuscirà mai a vedere come sono scritte le nostre pagine in PHP (questo per ragioni di sicurezza), quindi anche eventuali password o particolari istruzioni sono tenute al sicuro da occhi indiscreti.

Abbiamo visto come lavora la nostra prima istruzione in linguaggio PHP: l'istruzione **echo**. Questa istruzione equivalente alla istruzione **print**, in pratica stampa sul video la scritta che viene inserita tra le virgolette. Tra le virgolette può essere inserito un testo qualsiasi, però possono essere inseriti anche dei tag html come ad esempio il **break** (<br>) che esegue un ritorno a capo del testo. Si vede anche che ogni istruzione o rigo in linguaggio PHP deve essere terminato con il simbolo ; (*punto e virgola*).

Ad esempio, modificando l'istruzione di questo esempio in questo modo:

```
echo "Questo è il <b>primo programma</b> scritto in linguaggio PHP<br>";
```

ottego di avere a video una scritta in cui la parola “*primo programma*” viene scritta in grassetto (bold). Posso in questo modo, tramite i vari tag html, cambiare il font di caratteri, il colore, la dimensione, lo stile, ecc. ecc.

### Nota:

L'errore più comune nella scrittura delle pagine php è quello di *dimenticare* il *punto e virgola* a chiusura del rigo, si otterranno quindi output sul browser come questo

### Parse error: parse error in http://www.sito.it/www/ese01.php on line 18

occorre quindi aprire il file incriminato con l'editor, andare al rigo indicato e correggere il problema, spesso però l'errore si trova sul rigo precedente o ancora più indietro.

Un altro errore in cui si incappa spesso è quello di dimenticare i tag di apertura e chiusura del php, oppure di scrivere in maniera errata il tag di apertura `<?php` che non vuole spazi tra il *punto interrogativo* e la scritta *php*.

Passiamo quindi ad un altro esercizio, anche questo brevissimo ma che genera, come vedremo, un output gigantesco.

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

E' questa la prima **funzione** del linguaggio PHP che vediamo. Per **funzione** si intende un comando che riceve alcuni **parametri**, cioè alcuni dati, che vengono **passati** alla funzione stessa che poi li elaborerà. Il passaggio avviene inserendo questi parametri tra le parentesi tonde. I parametri possono essere 0 (nessuno, come in questo caso) oppure 1, 2, o molti di più come vedremo più avanti.

Il testo che compare tra le due linee sottostanti è relativo ad istruzioni PHP

PHP Version 5.2.8	
<b>System</b>	Windows NT LG-XP 5.1 build 2600
<b>Build Date</b>	Dec 8 2008 19:30:48
<b>Configure Command</b>	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-snapshot-template=d:\php-sdk\snap_5_2\vc6w8b\template" "--with-php-build=d:\php-sdk\snap_5_2\vc6w8b\php_build" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10sdk\shared"
<b>Server API</b>	Apache 2.0 Handler
<b>Virtual Directory Support</b>	enabled
<b>Configuration File (php.ini) Path</b>	C:\WINDOWS
<b>Loaded Configuration File</b>	C:\wamp\bin\apache\Apache2.2.11\bin\php.ini
<b>Scan this dir for additional .ini files</b>	(none)
<b>additional .ini files</b>	(none)

Il risultato è una pagina con decine di tabelle e centinaia di righe. Questa funzione viene utilizzata innanzitutto per testare se l'interprete PHP sul server funziona correttamente e per visualizzare le impostazioni di tutti i parametri del PHP stesso. Se andiamo a vedere il codice sorgente HTML della pagina che abbiamo ricevuto vedremo che è impressionante.

Passiamo al prossimo esempio, il nostro **terzo esercizio**.

Come ogni linguaggio di programmazione che si rispetti anche il PHP permette di far eseguire al computer delle operazioni matematiche. Cominciamo con quelle più semplici. Prima di partire però dobbiamo acquisire un nuovo concetto, quello di **variabile**. Semplificando al massimo possiamo dire che una variabile è una *locazione* o *cella di memoria* (RAM) in cui può essere scritto e da cui può essere riletto un valore qualsiasi.

Le **variabili** in **php** si indicano con il simbolo **\$** seguito da un nome qualsiasi. Il nome deve essere scritto utilizzando solo le lettere alfabetiche e le cifre e non può contenere simboli strani come le lettere accentate, i segni di interpunzione, lo spazio, ecc. Ad esempio una variabile si può chiamare **\$totale** oppure **\$quantita** (notare che ho usato la lettera a non accentata) oppure ancora **\$totale1** e **\$totale2**. Ricordarsi inoltre che il php è sensibile alle maiuscole-minuscole (case sensitive), per cui **\$Totale**, **\$totale** e **\$TOTALE** sono tre variabili diverse. Per semplificare ed evitare possibili errori utilizzerò sempre lettere minuscole.

```
19 <?php
20 $a = 10; $b = 3;
21 print "Se la variabile A vale: $a<br>";
22 print "Se la variabile B vale: $b<br>";
23 $result = $a+$b;
24 print "La loro somma vale: $result<br>";
25 $result = $a - $b;
26 print "La loro differenza vale: $result<br>";
27 $result = $a*$b;
28 print "Il loro prodotto vale: $result<br>";
29 $result = $a/$b;
30 print "La loro divisione vale: $result<br>";
31 $result = floor($a/$b);
32 print "Il quoziente vale: $result<br>";
33 $result = $a%$b;
34 print "Il modulo (resto) vale: $result<br>";
35 ?>
```

Analizziamo il listato (ho aggiunto per chiarezza il numero di riga in blu):

All'inizio vengono dichiarate due variabili (**\$a** e **\$b**) e vengono inizializzate cioè viene assegnato loro un valore iniziale (**riga 20**), notare che in un rigo possono stare anche due istruzioni php, purchè separate dal carattere punto e virgola;

viene stampata a video una scritta con del testo fisso ed una parte variabile, al posto del simbolo **\$a** il linguaggio php scriverà il valore che ha (*in quel momento*) la variabile **\$a** (**riga 21**);

la **riga 22** è simile al precedente ed è quindi inutile commentarlo;

nella **riga 23** viene eseguita una operazione matematica (la somma) tra due variabili ed il risultato di questa somma viene assegnato ad una terza variabile di nome **\$result** (la sequenza si legge da destra verso sinistra poiché il simbolo = (*uguale*) non ha il classico valore di uguaglianza come in matematica ma di **assegnazione**, cioè alla parte a sinistra viene *assegnato* il valore risultante dalla operazione che si trova a destra;

nella **riga 24** viene stampato a video il risultato di questa operazione;

le coppie di **righe 25-26**, **27-28** e **29-30** sono simili a quelle appena viste con la differenza che eseguono le altre tre operazioni fondamentali: differenza, prodotto e divisione;

nella **riga 31** troviamo una nuova funzione denominata **floor** (in inglese pavimento nel senso di basso) che esegue *l'arrotondamento verso il basso (per difetto)* del contenuto fra le parentesi tonde, ovvero viene eseguita prima la divisione e poi il risultato di questa viene arrotondato. Nello specifico 10/3 ha come risultato 3,33333 che poi viene arrotondato per difetto 3. E' questo il sistema che si può utilizzare per avere il quoziente di una divisione;

nella **riga 33** vediamo un altro operatore matematico un po' particolare, si chiama operazione di **modulo**, in pratica è una divisione di cui però viene preso il **resto**, infatti il risultato di questa operazione sarà 1 perchè 10/3 ha come quoziente 3 e resto 1.

Vediamo qui a lato un particolare dell'output visibile infine sul browser.

Chiaramente questi sono esempi molto semplici di cosa si può fare con un linguaggio di programmazione come il PHP che, puntualizziamolo ancora una volta non è un semplice linguaggio di descrizione della pagina

come può essere l'HTML, ma è un ben più potente strumento che ci permette di eseguire, cioè di fare alcune operazioni. Praticamente non esistono limiti su cosa si può fare con un linguaggio di programmazione. Fino ad ora abbiamo visto solo operazioni matematiche ma, come vedremo più avanti, si possono fare le cose più svariate, controllare se esistono file su disco, cancellarli, crearli, scriverli e sovrascriverli, controllare l'ora dell'orologio locale (del server) oppure quello remoto (del client) e in base a questi prendere delle decisioni, vedere da quale parte del mondo si collega il client, controllare l'accesso, tramite username e password, ad alcune risorse, controllare i dati su di un database per verificare la giacenza di un prodotto per indicare il prezzo di vendita, ecc. ecc.

Prima di passare ai due esercizi finale di questa prima lezione, parliamo ancora un attimo delle variabili. Le variabili avevamo detto poco fa sono delle celle di memoria in cui può essere contenuto un dato, un valore. Però i tipi di dati disponibili in questo linguaggio, come d'altronde in quasi tutti i linguaggi di programmazione, sono diversi, hanno cioè caratteristiche diverse.

I **tipi di dato** di cui avremo bisogno proseguendo in questo corso sono essenzialmente quattro:

**integer** (interi ovvero numeri senza decimali), **float** (fluttuanti ovvero numeri in virgola mobile), **boolean** (booleani ovvero binari con solo due valori possibili true e false) e **string** (stringhe ovvero sequenze di caratteri di testo).

Il tipo **integer** permette di utilizzare numeri compresi fra -128 e + 127 (usando 8 bit in memoria), oppure -32768 e +32767 (usando 16 bit in memoria) oppure ancora -2147483648 e +2147483648 (usando 32 bit). La codifica avviene in complemento a 2 per poter usare anche i numeri negativi ed in genere il numero di bit da utilizzare è automatico, cioè è completamente trasparente per l'utente che non se ne accorge. Ad esempio con l'istruzione **\$a = 317** viene creata in memoria una variabile di nome \$a che utilizzerà 16 bit ovvero due celle di memoria se lavoriamo con un microprocessore a 8 bit.

Il tipo **float** permette di utilizzare i *numeri reali*, cioè i *numeri in virgola mobile*. Ad esempio con l'istruzione **\$c = 123.45678** viene riservato in memoria una posizione per una variabile di nome \$c che conterrà il valore indicato. Da notare che il **separatore decimale è il punto**, essendo il linguaggio PHP di derivazione anglosassone. La codifica di questo numero in memoria avviene con meccanismi piuttosto particolari che non è qui il caso di approfondire, basterà accennare al fatto che la precisione di questi numeri è superiore alle dieci cifre significative e che vengono occupate diverse celle di memoria.

Il tipo **boolean** ovvero il tipo logico si può utilizzare per effettuare delle scelte quando si fanno dei confronti, il valore contenuto in memoria può essere solo **false** (*falso* in genere memorizzato come valore uguale a zero) e **true** (vero che viene memorizzato con un valore non zero). Un esempio di assegnazione di un valore booleano ad una variabile potrebbe essere questo: **\$aperto = true**.

Il testo che compare tra le due linee sottostanti è relativo

Se la variabile A vale: 10

Se la variabile B vale: 3

La loro somma vale: 13

La loro differenza vale: 7

Il loro prodotto vale: 30

La loro divisione vale: 3.33333333333333

Il quoziente vale: 3

Il modulo (resto) vale: 1

Questo invece è di nuovo testo in HTML.

L'ultimo tipo di base è il tipo **string** (stringa ovvero sequenza di caratteri) che serve per memorizzare le scritte o comunque dei dati alfanumerici. Un esempio di assegnazione è questo: **\$nome = "filippo"** che potrebbe essere anche indicato in questo modo **\$nome = 'filippo'**. L'uso dei *doppi apici o virgolette* al posto degli *apici singoli* è da preferire perchè all'interno delle virgolette potremo racchiudere, oltre al testo, anche altri simboli come ad esempio il nome di una variabile od altro come vedremo più avanti in questo corso.

Passiamo adesso al quarto esercizio che ci permetterà di prendere pratica con gli array (o vettori), che sono gruppi di variabili. L'uso dei vettori è importantissimo in qualsiasi linguaggio di programmazione poiché permette, utilizzando pochissime istruzioni, di effettuare lunghissime sequenze di operazioni con la scrittura di poche righe di codice.

Ricordatevi sempre che più righe scriverete maggiore sarà la probabilità di commettere degli errori, inoltre dovendo fare manutenzione al proprio codice, a distanza di tempo, il codice stesso diventerà di difficile lettura.

Ad esempio supponiamo di memorizzare il nome delle 10 province della Toscana e di stampare su video il loro nome, vediamo quindi due approcci completamente diversi allo stesso problema, uno concettualmente semplice, ma idiota, ripetendo a pappagallo tante righe, ed uno più intelligente, anche se a prima vista può sembrare inutilmente più complicato, ma molto più flessibile.

```
<?php
    $toscana1 = "Arezzo";
    $toscana2 = "Massa Carrara";
    $toscana3 = "Firenze";
    $toscana4 = "Grosseto";
    $toscana5 = "Livorno";
    $toscana6 = "Lucca";
    $toscana7 = "Pisa";
    $toscana8 = "Pistoia";
    $toscana9 = "Prato";
    $toscana10 = "Siena";
    echo "Le province della Toscana sono:<br><i>";
    echo "$toscana1<br>";
    echo "$toscana2<br>";
    echo "$toscana3<br>";
    echo "$toscana4<br>";
    echo "$toscana5<br>";
    echo "$toscana6<br>";
    echo "$toscana7<br>";
    echo "$toscana8<br>";
    echo "$toscana9<br>";
    echo "$toscana10<br></i>";
?>
```

In pratica ho creato 10 variabili diverse, ho assegnato ad ognuna di esse un valore corrispondente al nome di una provincia e poi ho stampato con 10 istruzioni il contenuto delle 10 variabili.

Il risultato sul browser è quello visibile qui a lato.

Il problema arriva quando, ad esempio, si vogliono memorizzare e stampare le cento ed oltre province italiane. Non possiamo scrivere quintali e quintali di codice, occorre trovare una soluzione più elegante. Diventerà molto difficile gestire centinaia di variabili con nomi diversi, qualche errore lo combineremo sicuramente, possiamo facilmente confonderci con i nomi delle variabili, anche perchè probabilmente utilizzeremo la funzione di copia e incolla dell'editor, modificando poi in seguito solo alcuni caratteri, come ho fatto io con l'esempio indicato qui

Il testo che compare tra le due linee

```
Le province della Toscana sono:
Arezzo
Massa Carrara
Firenze
Grosseto
Livorno
Lucca
Pisa
Pistoia
Prato
Siena
```

Questo invece è di nuovo testo in H

sopra. E se mi dimentico di modificare qualcosa? Trovare un errore logico è veramente difficile, mentre un errore di sintassi viene prontamente segnalato dall'interprete php che, anzi, mi indica anche in quale riga si trova. Mentre tra centinaia di righe accorgersi che la variabile \$provincia77 compare due volte mentre manca la variabile \$provincia78 perchè mi sono dimenticato di correggere il testo dopo il copia ed incolla è quasi impossibile!

La stessa cosa poteva essere fatta utilizzando gli array, ovvero un metodo più intelligente. In questo caso la variabile è unica ovvero una sola però che occupa diverse locazioni in memoria. Nell'esempio seguente alla variabile \$toscana, definita come array ovvero vettore ovvero insieme di variabili, vengono assegnati 10 valori diversi.

Per accedere al primo valore del vettore indicherò \$toscana[0], per accedere al secondo valore del vettore indicherò \$toscana[1] e così via. Il valore indicato tra parentesi quadre si chiama indice e parte sempre da zero. Vediamo il codice risultante:

```
<?php
    $toscana = array ("Arezzo","Massa Carrara","Firenze","Grosseto","Livorno",
                    "Lucca","Pisa","Pistoia","Prato","Siena");
    echo "Le province della Toscana sono:<br><i>";
    echo "$toscana[0]<br>";
    echo "$toscana[1]<br>";
    echo "$toscana[2]<br>";
    echo "$toscana[3]<br>";
    echo "$toscana[4]<br>";
    echo "$toscana[5]<br>";
    echo "$toscana[6]<br>";
    echo "$toscana[7]<br>";
    echo "$toscana[8]<br>";
    echo "$toscana[9]<br></i>";
?>
```

Il risultato sul browser è identico, però qualcuno di voi potrebbe obiettare: *“Sì, però cosa abbiamo guadagnato, abbiamo risparmiato solo alcune righe ed inoltre è lo stesso facile sbagliare il numerino tra le parentesi quadre...”*.

In effetti ancora non si vedono i grandi vantaggi, però modificando leggermente il codice qualcosa si comincia ad intravedere...

```
<?php
    $toscana = array ("Arezzo","Massa Carrara","Firenze","Grosseto","Livorno",
                    "Lucca","Pisa","Pistoia","Prato","Siena");
    echo "Le province della Toscana sono:<br><i>";
    $i = 0;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br>"; $i = $i + 1;
    echo "$toscana[$i]<br></i>";
?>
```

Se aggiungo una variabile, ad esempio \$i, che posso utilizzare come indice del vettore, inizializzandola a zero ed incrementandola di uno ad ogni riga che viene stampato ottengo lo stesso risultato in maniera molto più elegante e, soprattutto, riducendo la possibilità di commettere errori poiché tutte le righe sono identiche tra loro.

Cominciate a vedere le potenzialità dell'uso di un linguaggio di programmazione? Spero proprio di sì, perché guardate come si accorcia notevolmente il listato utilizzando un altro tipo di istruzione del

linguaggio php che ancora non potremo conoscere e che vedremo ed utilizzeremo nel corso della prossima lezione. Ed il risultato visualizzato nella pagina web vista dal client è il medesimo, indistinguibile dagli altri esempi. Inoltre questo ultimo esempio rimane praticamente così corto anche se volessi stampare tutte le province d'Italia.

```
<?php
$toscana = array("Arezzo","Massa Carrara", "Firenze","Grosseto","Livorno",
                "Lucca","Pisa","Pistoia","Prato","Siena");
echo "Le province della Toscana sono:<br><i>";
for( $i = 0; $i < 10; $i = $i + 1 ) {
    echo "$toscana[$i]<br>";
}
echo "</i>";
?>
```

Con il quinto esercizio proviamo a lavorare un po' con le date. La data in php viene in realtà memorizzata come un numero, un po' come avviene nel tabellone elettronico Excel, questo ci permette di eseguire con le date alcuni semplici calcoli.

Ad esempio, se \$oggi contiene una data, la data odierna, cosa si ottiene con l'operazione **\$oggi+1**? Si ottiene la data di **domani**. Mentre con **\$oggi-1** si ottiene la data di **ieri**, con **\$oggi-365** si ottiene la data odierna però di un anno fa e così via.

Proviamo subito con un altro esercizio:

```
<?php
echo date("d/m/y")."<br>";
echo date("d/m/Y")."<br>";
?>
```

Il testo che compare tra i

01/02/09  
01/02/2009

Guardando la pagina web si vede che è stata visualizzata la data odierna e cioè il 1° febbraio 2009 in due modi diversi, con l'anno espresso con due cifre e con quattro cifre.

Questo invece è di nuovo

Per visualizzare la data ho utilizzato una funzione php e cioè la funzione **date( )** che prevede che vengano inseriti uno o due parametri all'interno delle parentesi tonde. Il primo rappresenta il formato con cui si vuole visualizzare la data, mentre il secondo è la data da visualizzare, se viene omesso il secondo parametro viene utilizzata la data odierna.

Per quello che riguarda il formato sono validi i seguenti simboli:

**d** = day (giorno del mese) con 2 cifre,

**j** = giorno senza zero anteposto ossia con 1 o 2 cifre,

**l** = day (elle minuscola) scritto all'inglese al completo (Friday, Saturday, Sunday, Monday, ecc.),

**D** = day (giorno del mese) però scritto abbreviato all'inglese con 3 lettere (Fri, Sat, Sun, Mon, ecc.),

**m** = month (mese) con 2 cifre,

**n** = mese senza zero anteposto ossia con 1 o 2 cifre,

**F** = mese però scritto all'inglese al completo (January, February, March, ecc.),

**M** = mese però scritto abbreviato all'inglese con 3 lettere (Jan, Feb, Mar, ecc.),

**y** = year (anno) con 2 cifre cioè senza secolo e millennio,

**Y** = year però con 4 cifre,

**H** = hour (ora) con il formato a 12 ore (am/pm),

**h** = hour però con il formato a 24 ore,

**i** = minutes (minuti),

**s** = seconds (secondi).

Vediamo di seguito un altro esempio ed il relativo output ottenuto sul browser.

<pre>&lt;?php echo date("d/m/Y")."&lt;br&gt;"; echo date("d/m/Y")."&lt;br&gt;"; echo date("D d F Y")."&lt;br&gt;"; echo date("H:i:s")."&lt;br&gt;"; ?&gt;</pre>	<p style="color: red; font-weight: bold;">Il testo che compare tra le due linee s</p> <hr/> <p>01/02/09 01/02/2009 Sun 01 February 2009 23:46:55</p> <hr/>
---	--

Possiamo quindi scrivere le date e gli orari, che in realtà fanno parte della data stessa, in vari modi.

Abbiamo notato che, come i primi sistemi operativi e tutti i linguaggi di programmazione che si esprimono in questa lingua *universale* del programmatore, anche nel linguaggio PHP le date vengono espresse con la notazione anglosassone. Vedremo in seguito come fare per esprimerle in italiano o, comunque, in qualsiasi altro linguaggio.

Introduciamo adesso un altro modo attraverso il quale è possibile esprimersi manipolando le date. Esiste una funzione del php che serve per creare una data, questa funzione si chiama **mktime( )** ovvero **maketime** e si usa ad esempio in questo modo **\$filippo = mktime( 1,30,0,11,20,1989,0)**, la sintassi potrebbe essere riassunta in questo modo

**\$data = mktime ( <ore>,<minuti>,<secondi>,<mese>,<giorno>,<anno>,<oralegale> );**

dove ore è un numero compreso tra 0 e 23, minuti tra 0 e 59, ecc. ecc.

per *oralegale* si indica **o (false)** per ora solare oppure **-1 (true)** per ora legale in corso.

<pre>echo "Altro modo di lavorare con le date&lt;br&gt;"; \$data1 = mktime( 12,30,0,2,10,1969,0); echo "La data indicata è: \$data1&lt;br&gt;"; \$data2 = mktime( 12,30,0,2,10,1971,0); echo "La data indicata è: \$data2&lt;br&gt;";</pre>	<p style="color: red; font-weight: bold;">Il testo che compare tra le due linee sottostanti</p> <hr/> <p>Altro modo di lavorare con le date La data indicata è: -28038600 La data indicata è: 35033400</p> <hr/> <p style="color: red; font-weight: bold;">Questo invece è di nuovo testo in HTML.</p>
---	--

Cosa è successo? Cosa sono questi numeri? Semplice e complicato nello stesso momento: il numero rappresenta i secondi trascorsi da un riferimento, tutte le date vengono memorizzate con il sistema *Unix-Like*, ovvero come numero di secondi trascorsi dalla mezzanotte del 1° gennaio 1970, quindi la data riferita al 1969 sarà rappresentata da un valore negativo, mentre la data riferita al 1971 sarà rappresentata da un valore positivo. Un po' come il nostro modo di indicare le date che parte dalla nascita di Cristo utilizzata come riferimento, per gli ebraici e per i musulmani il riferimento è diverso.

Agli effetti pratici non cambia niente, perchè in genere si lavora con differenza tra le date e quindi il futuro avrà sempre valori maggiori (ovvero positivi) rispetto ad oggi, ed il passato avrà valori minori (negativi) rispetto ad oggi. In ogni caso la funzione **date( )** che abbiamo già visto riporta questi valori in scritte umanamente comprensibili e la funzione **mktime( )** provvede a memorizzare correttamente nel formato Unix le date che dovremo inserire.

Attenzione solo al fatto che la funzione **mktime( )** ha il mese ed il giorno scambiati di posizione rispetto alla notazione italiana, come al solito parliamo di un linguaggio di derivazione anglosassone.

Con questo esempio finale concludiamo questa lezione.

```
echo "Altro modo di lavorare con le date<br>";
$filippo = mktime( 1,30,0,11,20,1989,0);
$adesso = mktime( 13,10,0,1,24,2009,0);
echo "Filippo è nato il: $filippo (secondi a partire dal 1/1/1970)<br>";
echo "Filippo è nato il: ".date("D d/m/Y",$filippo)."<br>";
$eta = ($adesso - $filippo)/(60*60*24);
echo "L'eta' di Filippo è di: $eta (giorni)<br>";
```

In pratica ho definito una variabile di nome **\$filippo** che contiene la data di nascita di una persona, inserita utilizzando la funzione `mktime()`; si suppone che questa persona sia nata alle **ore 01:30:00 del 20/11/1989** e si suppone anche che la data/ora attuale sia indicata con la variabile **\$adesso** e corrisponda alle **ore 13:10:00 del giorno 24/01/2009**.

Le istruzioni indicate faranno visualizzare la data/ora di nascita di Filippo espressa in notazione Unix (ovvero secondi a partire dal 1° gennaio 1970) e in notazione normale, inoltre viene calcolata anche l'età di Filippo, esprimendola però in giorni (dividendo il numero di secondi per  $60 \times 60 \times 24 = 86400$ ).

Ecco il risultato visualizzato sul browser.

Il testo che compare tra le due linee sottostanti è relativo ad istruzioni di shell.

```
Altro modo di lavorare con le date
Filippo è nato il: 627525000 (secondi a partire dal 1/1/1970)
Filippo è nato il: Mon 20/11/1989
L'età di Filippo è di: 7005.4861111111 (giorni)
```

Questo invece è di nuovo testo in HTML.

L'ultima cosa da notare è l'uso dell'operatore `.` (*punto*) che serve a congiungere due o più stringhe di testo con il medesimo comando **echo**.

Ad esempio:

```
echo "Buongiorno"."Signore";
```

visualizzerà a video la scritta BuongiornoSignore, notare che non ci sono spazi tra le due scritte, se volevo separarle occorrerà aggiungere esplicitamente uno spazio, ad esempio in questo modo:

```
echo "Buongiorno"." ". "Signore";
```

Ho ritenuto di dover puntualizzare meglio questo perché l'errore dovuto alla dimenticanza del punto o dello spazio tra le virgolette è molto frequente ed è subdolo da eliminare.

La stessa cosa poteva essere ottenuta mettendo le istruzioni su più righe:

```
echo "Buongiorno";
echo " ";
echo "Signore";
```

Avrebbe provocato lo stesso output su video, visto che non è presente il tag `<br>` che provoca il ritorno a capo della riga di testo. In mancanza del tag `<br>` la scrittura prosegue sul rigo corrente fino al limite massimo a destra e solo dopo la scrittura prosegue a capo, se però viene ridimensionata la finestra del browser il testo si ridispone nel nuovo spazio disponibile.