Introduzione: linguaggio PHP

> **PHP** (acronimo ricorsivo che sta per "*PHP: Hypertext Preprocessor*" cioè PHP: preprocessore di ipertesti) è un **linguaggio di scripting general-purpose Open Source** molto usato, specialmente **indicato per lo sviluppo Web** e può essere integrato nell'HTML.

PHP è dunque un prodotto *open source* e gratuito, un linguaggio di scripting interpretato comprensibile dalla maggior parte dei browser; proprio per questi motivi assume sempre più una larga diffusione e tende a soppiantare il classico e statico HTML. Originariamente concepito per la realizzazione di pagine web dinamiche, attualmente è utilizzato principalmente per sviluppare applicazioni web lato server ma può essere usato anche per scrivere script a linea di comando o applicazioni standalone con interfaccia grafica.

Uno dei suoi concorrenti principali è ASP, tecnologia creata dalla Microsoft. Una delle differenze fondamentali è che ASP può essere letto solo da browser Microsoft (Internet Explorer). In origine ASP assumeva un aspetto privilegiato solo perché la velocità in esecuzione era superiore rispetto al concorrente. Dall'uscita della versione PHP4 questa differenza è notevolmente diminuita se non superata per certi aspetti. Al momento l'ultima versione in circolazione è PHP5.

Il linguaggio PHP è un linguaggio che deposita sul server file che, quando vengono richiamati, generano *al volo* pagine in codice HTML (ma anche JavaScript, CSS ecc..) caricate sul browser dell'utente.

PHP è un linguaggio di programmazione che permette di creare ***pagine dinamiche***, tramite l'aiuto di database esterni o il passaggio dati fra le stesse pagine. Una delle caratteristiche più importanti e significative di PHP è infatti la possibilità di **supportare una completa gamma di databases** (MySql, PostgreSql, Sql Server, Oracle, SyBase, Access e altri).

Con PHP si ha la libertà di scegliere **praticamente qualsiasi sistema operativo** e supporta la **maggior parte dei server web esistenti**. Php proviene come concetti di base dal linguaggio C.

Una delle sue grosse potenzialità è la semplicità di apprendimento e **l'elasticità delle procedure**.

Si può scegliere se fare uso di una **programmazione procedurale** oppure **orientata agli oggetti**, o una combinazione di entrambe: con la version 5, il PHP supera le debolezze dell'OOP presenti in PHP 4 (dove non erano realizzate tutte le caratteristiche standard di OOP) ed introduce in modo completo il modello a oggetti.

Esistono **tre campi principali** in cui vengono usati gli scripts PHP.

- Lo **scripting server-side**. Questo è il campo più tradizionale ed il maggiore obiettivo del PHP. Per fare questo lavoro occorrono tre cose: il parser PHP (CGI o server module), un webserver ed un browser web.
- Lo **scripting di righe di comando**. Si può creare uno script PHP da usare senza alcun server o browser. Per usarlo in questo modo, l'unica cosa necessaria è un parser PHP. Questi script possono essere utilizzati per semplici task di processamento testi.
- Scrittura di **applicazioni desktop**. Probabilmente PHP non è il linguaggio più adatto per scrivere applicazioni desktop, con interfaccia grafica, ma, se se ne vogliono usare delle caratteristiche avanzate in applicazioni client-side, si può anche adoperare PHP-GTK (estensione non reperibile nella grande distribuzione) per scrivere questo tipo di programmi o scrivere applicazioni cross-platform.

Le possibilità di PHP, oltre all' output in HTML, includono l'abilità di generare immagini, files PDF e perfino filmati Flash al volo (utilizzando libswf e Ming). Si può generare facilmente qualsiasi testo, come XHTML e qualsiasi altro file XML. PHP può autogenerare questi file, e salvarli nel file system, piuttosto che eseguire un printing esterno, o creare server-side cache per contenuti dinamici.

Struttura del linguaggio PHP:

Premesse:

1) Ogni istruzione PHP deve finire con il ***punto e virgola***.

2) Per scrivere a video le parole riservate dobbiamo inserire la ***barra*** \ prima della parola.

Istruzioni iniziali:

Il linguaggio PHP prevede come in ogni linguaggio un inizio ed una fine.

I comandi che lo regolano sono i seguenti:

<?php //tag di apertura

....Codice PHP....

?> //tag di chiusura

Tutto quello racchiuso fra queste due istruzioni non sarà mai visibile dall'utente.

Un codice alternativo può essere:

<script language="php">
... istruzioni...
</script>

Ci sono altri modi per aprire e chiudere il codice PHP (ad esempio il codice può essere inserito tra < % e %>), ma per usarli dobbiamo settare i parametri di configurazione.

Esistono 4 set di tag che possono essere utilizzati per delimitare blocchi di codice PHP. Solo i due già citati (<?php. . .?> e <script language="php">. . .</script>) sono sempre disponibili. Gli altri due sono i **tag brevi** e i **tag stile ASP** e possono essere attivati o disattivati tramite il file di configurazione *php.ini*. Sebbene i tag brevi o quelli in stile ASP possano essere pratici, questi sono meno portabili e, in generale, sconsigliati.

```
1.  <?php echo 'per produrre documenti XHTML o XML, usare questo modo'; ?>
2.  <script language="php">
        echo 'alcuni editor non amano le istruzioni di elaborazione';
    </script>

3.  <? echo 'questo è il più semplice, ovvero come istruzione SGML'; ?>
    <?= espressione ?>  Questa è un'abbreviazione per "<? echo espressione ?>"


4.  <% echo 'Opzionalmente puoi utilizzare tag nello stile ASP'; %>
    <%= $variable; # Questo &egrave; una abbreviazione per "<% echo ..." %>
```

Nota: Occorre notare che se si intende inserire codice PHP all'interno di testi XMl o XHTML, occorre utilizzare **<?php ?>** per essere conformi allo standard XML.

Le istruzioni in PHP possono essere in qualunque parte della pagina e si possono integrare con HTML.

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

In questo caso il server elabora solo il codice PHP lasciando invariato il codice HTML.

Restituire codice:

I due **costrutti nativi** o istruzioni che permettono di scrivere il codice utente (Html, JavaScript, css, ecc..) sono:

print();
echo();

La sintassi può variare anche in: *print"..testo..";*
echo"..testo..";

All'interno possiamo inserire stringhe o istruzioni (codice compreso fra apici doppi o singoli se non è codice PHP).

Questi due comandi permettono di creare le pagine che verranno caricate sul browser dell'utente. Il comando echo a differenza di print può contenere più stringhe, purché siano separate da una virgola.

Esempio:
```
echo"Ciao!","<br>\n","Come state?";      // più stringhe
print"Ciao!"."<br>\n"."Come state?";      // punto per concatenare
```

Come abbiamo appena visto possiamo introdurre sia frasi che istruzioni html. La cosa importante è il segno \n dopo il tag BR. Serve esclusivamente per impaginare in modo corretto il documento html che risulterà dal codice php. Senza \n tutto il codice html ottenuto da php genererà una sola riga, rendendo problematica la sua lettura dal programmatore.

Commenti:

I commenti sono simili ad altri linguaggi:

//	Per commenti su una sola riga.
#	Per commenti su una sola riga.
/*.....*/	Per commenti su più righe.

Esempio:
```
<?php
        $a="tutti";
        echo"<b>Ciao a ".$a."</b>";          // $ indica una variabile
 ?>
```

In questo caso nella pagina che arriva nel browser troveremo scritto: *ciao a tutti*
e l'utente potrà vedere a video: **ciao a tutti**

In pratica abbiamo inserito del codice HTML con il linguaggio PHP. Questo è quello che crea le pagine dinamiche.

Immaginiamo di avere un database con 100 dati salvati, e li vogliamo scrivere dentro una tabella.

Con Html dobbiamo scrivere 100

<*tr*><*td*>dato</*td*></*tr*>

Con php ci connettiamo al database, inseriamo un ciclo if ed abbiamo scritto la tabella! Senza considerare che ogni volta che cambiamo la tabella non dobbiamo cambiare il codice nella pagina PHP, perchè avviene in automatico!

Posizione:

Il codice PHP può essere inserito in qualunque parte della pagina.
Se prendiamo l'esempio sopra possiamo scrivere:

```
<table border="1">
<?php
        echo"<tr><td>ciao a tutti</td></tr>";
?>
</table>
```

Risultato:

ciao a tutti

In pratica una pagina in PHP è una pagina con costrutti di elaborazione, codice e funzioni.

Una pagina in PHP di solito è una pagina con codice HTML e PHP, dove il server legge solo il codice PHP, lo elabora e lo spedisce al browser sotto forma di linguaggio HTML.

Si possono avere delle eccezioni a quello detto sopra se il codice viene inserito tramite ***include*** o ***require*** ossia quando una pagina viene formata da più pagine unite insieme (un esempio simile sono i fogli di style richiamati quando necessario).

Variabili:

Come ogni linguaggio che si rispetti anche PHP mette a disposizione le variabili.

La sintassi per **inizializzare** ed **assegnare** dei valori è la seguente:

$a = 5;

Per stampare un valore risultante da una variabile basta scrivere:

print $a; // Senza inserire apici.

Le variabili dovranno sempre essere inserite precedute dal segno dollaro ***$***

Importante: le variabili in PHP non necessitano di essere inizializzate prima del loro effettivo utilizzo.

Tipi di variabili: http://docs.php.net/manual/it/language.types.php

- **INTERE**

Quando la variabile viene associata ad un valore intero, positivo o negativo. Il PHP accetta anche altri valori ad esempio ottali o esadecimali.

Esempio: ***$a=5;***
$b=5*10;

- **VIRGOLA MOBILE**

Quando la variabile è associata ad un valore con virgola (In PHP si usa il punto al posto della virgola!!!).

Esempio: ***$a=5.5;***

Il numero di valori dopo il punto può essere variato tramite il parametro '*precision*' nel file *php.ini* (in locale).

- **BOOLEAN**

Quando la variabile assume il valore 'true' o 'false'.

Esempio: ***$a=true;***

- STRINGHE

Quando la variabile assume il valore di una stringa. La variabile stringa può essere utilizzata in tre modi:

1) Con **apici semplici**.

Questo metodo permette di scrivere esattamente uguale a quello che vi è all'interno, ma così non riconosciamo le variabili.
Esempio:

```
$a='voi';
print 'saluto tutti $a';        // Otterremo come risultato 'saluto tutti $a'
```

2) Con **apici doppi**.

Con questo metodo possiamo riconoscere le variabili contenute nella stringa (*PHP risolve* le variabili).
Esempio:

```
$a='voi';
print "saluto tutti $a";      // Otterremo come risultato 'saluto tutti voi'
```

3) **EOD** cioè un insieme di tre lettere che delimitano il commento su più righe

Con questo metodo possiamo riconoscere le variabili contenute nella stringa.
La particolare sintassi prevede che la stringa inizi con **<<<*EOD*** (anche <<< STR) seguito dalla stringa. Alla fine dobbiamo andare a capo e scrivere ***EOD;*** (o STR;)
Esempio:

```
$a='voi';
$b= <<<EOD saluto tutti $a
EOD;
print $b;            // Otterremo come risultato 'saluto tutti voi'
```

Per inserire delle parole riservate dobbiamo usare \

Esempio: ***print 'Ciao un\'altra volta';*** // Otterremo come risultato '*Ciao un'altra volta*'

Altri esempi[1].

Conversione da stringa a intero

Esempi per convertire da stringa a valore intero:

- Usare l'operatore di **concatenazione**
  $num= 0 + "5 rose ... i numeri seguenti non considerati ... 2" ;

- Usare l'operatore di **cast**
  $s="5";
  $num2=(**int**)$s;
  $num3=(i**nteger**)$s;

- Usare,dalla versione 3, la funzione intval (mixed $var [,int $base]) con base per default decimale
  $num1= **intval** ($s);

Cancellare variabili:

Se per qualunque motivo dobbiamo cancellare una variabile dobbiamo usare la ***funzione unset().***

Sintassi: ***unset($a);***

Variabili globali e locali:

Le variabili ***globali*** sono le variabili definite all'esterno di funzioni, nello script principale e che possono essere utilizzate da tutto il programma PHP escluso all'interno di funzioni. Viceversa per variabili ***locali*** intendiamo solo quelle contenute all'interno di funzioni, che non potranno essere usate all'esterno delle stesse.

Detto questo se immaginiamo una situazione dove due variabili hanno lo stesso nome ma sono diverse fra loro ossia una locale e l'altra globale, possiamo stare tranquilli che l'una non influenzerà l'altra e viceversa.

Esempio:

```
$a = 5;          // variabili globali
$c = abc();
function abc(){
        $a = 10+3;        // variabile locale
        print $a."\n";
}
print $a;
```

Scriveremo:
13
5

Le due variabili sono distinte fra loro e con due valori ben diversi.

[1] Uso di funzioni PHP per gestire le stringhe

Global:

Si può verificare un'eccezione alla regola globale/locale se per qualsiasi motivo vogliamo usare direttamente una variabile globale all'interno di una funzione.
Il problema si risolve con il comando *global*.
Esempio:

```
$a = 5;
$c = abc();
function abc(){
        global $a;          // uso della variabile globale
        print $a."\n";
}
print $a;
```

Scriveremo:
5
5

Riferimento a variabili:

Come nel linguaggio C possiamo riferirci alle variabili. Il problema nasce nel momento in cui **manipoliamo una variabile globale in una funzione e vogliamo che cambi valore anche esternamente**. Risolviamo il tutto richiamando la funzione non con il nome della variabile, ma con il suo riferimento al nome.

La sintassi prevede di dichiarare la variabile fra le proprietà della funzione con il simbolo ***&*** (e commerciale) che la precede.

Esempio:

```
$a = 5;
print $a."<br>";
$c = abc($a);
function abc(&$a){
$a++;
print $a."<br>";
}
print $a."<br>";
```

Scriveremo:
5
6
6

Da adesso il valore della variabile globale resterà variato con il nuovo.

Varibili resource:

Le variabili resource sono le variabili risultanti di una connessione esterna, solitamente su **database**. Queste variabili non possono essere manipolate (non possono variare) e non le possiamo usare per nessun calcolo.

Il loro unico utilizzo è quello di ***puntare ad una connessione ben precisa***. Tutte le volte che verrà usata sapremo che quella funzione si riferisce a quella precisa connessione.

Sintassi per inizializzarla:

$nomeacaso = ...funzione che apre la connessione...;

Da adesso tutte le funzioni che si riferiscono a quella connessione dovranno avere all'interno delle parentesi tonde la variabile resource.

Esempio: ***mysql_select_db('nomedatabase', $nomeacaso);***

Alcune query[2]:

Inserire riga

mysql_db_query($db_database,"**insert into** ".$db_tabella."(nome, email) **values**('".$nome."', '".$email."')",$xxx);

Eliminare riga

mysql_db_query($db_database,"**delete from** ".$db_tabella." **where** id=5 and....",$xxx);

Modificare riga

mysql_db_query($db_database,"**update** ".$db_tabella." **set** nome='".$nome."', email='".$email."' **where** id=5 and....",$xxx);

Interrogare tabella

mysql_db_query($db_database,"**select** nome, email **from** ".$db_tabella." **where** id=5 ",$xxx);

ATTENZIONE: Nel caso che all'interno di una query si utilizzi una variabile che contiene una stringa si deve operare nel seguente modo:

Esempio:

mysql_db_query($db_database,"select nome, email from ".$db_tabella." where cognome='" ***.$cogn."'*** *",$xxx);*

Come avrete notato la variabile **$cogn** è contenuta da doppi apici e apici: **' " $cogn " '**

Questo perchè i doppi apici contengono la variabile, che una volta sostituita dal testo string lasciano il posto agli apici (per contenere il testo della stringa che altrimenti non verrebbe distinta dal testo query e genera un errore).

---

[2] Interazione PHP con il database relazionale MySQL

PHP: linguaggio server side nella creazione di pagine web dinamiche

Get

I metodi Get e Post non sono altro che i modi con cui possiamo far spedire i dati dall'utente al server o in generale scambiare informazioni tra due pagine HTML. Fondamentali per conferire al sito un aspetto dinamico e funzionale.

Il metodo *get* prevede che i nomi ed i valori da associare siano posti direttamente sul link che richiama la pagina.

La sintassi è la seguente: ***<a href="nome_pagina.php?nome=valore>***

Come vediamo dall'esempio basta ricordarsi del *punto interrogativo* e non sbagliare il nome o il valore della variabile...

Se vogliamo inserire più nomi e valori, dobbiamo solo inserire il carattere **&** (e commerciale, che significa AND) fra le coppie di parametri.

Esempio: ***<a href="nome_pagina.php?nome=valore&nome2=valore2&nome3=valore3....>***

Dopo aver inserito questa riga (che per convenzione si chiama **query string**) abbiamo a disposizione un array ***superglobale*** (cioè utilizzabile anche all'interno di funzioni senza doverlo dichiarare *global*) di nome ***$_GET*** che contiene tutti i parametri che abbiamo inserito nella query string ed è visibile anche dalla nuova pagina.

Esempio dell'array: ***$_GET = array('nome'=>'valore', 'nome2'=>'valore2', 'nome3'=>'valore3');***

Da adesso abbiamo a disposizione tutte le variabili che contiene l'array associativo:

$_GET['nome']; // che equivale a *'valore'*
$_GET['nome2']; // che equivale a *'valore2'*
$_GET['nome3']; // che equivale a *'valore3'*

Esempio:
Supponiamo di scrivere una riga come questa: ***<a href="index.php?ciao=2&ciao2=4>***

Quando il codice arriva al server abbiamo a disposizione le variabili:

$ciao=2;
$ciao2=4;

E le possiamo usare per generare la pagina index.php che vedrà l'utente sul suo browser.

Sicurezza:

Per motivi di sicurezza il server imposta ***conf_global*** su *OFF* e non possiamo usare le variabili con il loro nome se non dopo averle dichiarate nel seguente modo (nella pagina di destinazione, nel nostro esempio *index.php*):

$ciao=$_GET['ciao'];
$ciao2=$_GET['ciao2']; *// a*desso possiamo usare le due variabili **$ciao=2** e **$ciao2=4**

Conclusioni:

Le query string sono una delle caratteristiche fondamentali dei linguaggi che creano pagine dinamiche.
I dati scelti dall'utente (tramite il form di ricerca) vengono rimbalzati dalla pagina iniziale al server, analizzati, manipolati e poi di nuovo visualizzati sul browser.
In questo modo è possibile generare migliaia di soluzioni per la stessa pagina web, creando siti dinamici.

NOTA BENE:

In alcuni vecchi browser il carattere **& (e-commerciale)** non è previsto nel codice HTML (è una *parola riservata*), ed il navigatore potrebbe avere dei problemi quando visualizza una pagina dove è presente una query che trasmette più variabili con metodo get.
Per ovviare a questo problema è sempre consigliabile scrivere il carattere **&** con il corrispettivo codice HTML:

&

Esempio:

Scriviamo sull'editor... ***<a href="index.php?ciao=2&ciao2=4>***

...e vediamo nella barra indirizzi ***<a href="index.php?ciao=2&ciao2=4>***

Post

E' preferibile usare il metodo Post come modalità per spedire i dati dall'utente al server quando si voglia privilegiare **sicurezza** e non essere limitati nella dimensione dell'invio. Il metodo Get infatti, è limitato in quanto la url non può contenere più di 255 caratteri ma specialmente è poco sicuro: se si usasse, supponendo di inviare *nome_utente* e *nome_psw* inseriti nel form, nella pagina successiva si avrebbe una url di questo tipo: *nome_pagina_php*.php?nome= *nome_utente*&password= *nome_psw* che rende visibili i valori dei campi "nome" e "password" che solitamente si vogliono nascondere.

Il metodo post si usa con i <form>, non accoda le variabili alla url della pagina e crea un array associativo che si chiama ***$_POST***. Con il metodo post si creano tante variabili quanti sono i dati inseriti nel form.

L'array ***superglobale*** generato conterrà delle chiavi che portano il nome (***name***) delle caselle del form ed i valori saranno quelli che l'utente avrà inserito.
.

Esempio:
```
<form action="prova.php" method="post">
    <input type="text" name="nome1">
    <input type="checkbox" name="nome2" value="si">
    <input type="submit" name="submit" value="invia">
</form>
```

Da adesso abbiamo a disposizione l'array $_POST:

$_POST = array('nome1'=>'simone', 'nome2'=>'si');

Se la casella "checkbox" non viene cliccata la variabile non verrà definita.

Sicurezza:

Per motivi di sicurezza il server imposta ***conf_global*** su *OFF* e non possiamo usare le variabili con il loro nome se non dopo averle dichiarate nel seguente modo (nella pagina di destinazione del form):

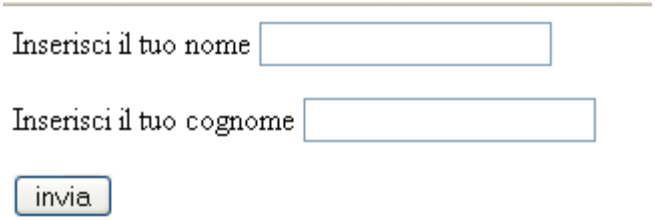
$var=$_POST['var'];
$var2=$_POST['var2'];

Adesso possiamo usare le due variabili **$var** e **$var2**

Esempio di creazione di pagine web dinamiche

- **Scopo**: realizzare moduli HTML per l'invio dei dati attraverso metodi GET o POST e pagine PHP in grado di ***riceverli***, ***interpretarli*** e ***maneggiarli*** al fine di produrre un output dimostrativo delle potenzialità di tale linguaggio.

Esercizio: creare un **form** (scheda) di inserimento dati salvando il documento con nome **ins.htm** (che invia dati con metodo **get**) o **ins1.htm** (che invia dati con metodo **post**) che si vuole con il seguente layout:



- Scrivere nel file con nome **reg.php** (in risposta all'invio con metodo **get)** o **reg1.php** (in risposta all'invio con metodo **post**) il programma che elabora i dati della scheda visualizzando sulla finestra del browser un messaggio di benvenuto personalizzando nome e cognome

Soluzione

Codice pagina html con uso metodo GET:

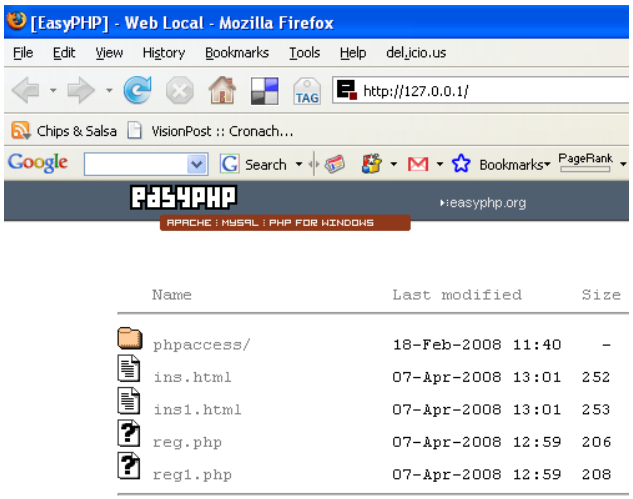
```
<form action="reg.php" method="get">
Inserisci il tuo nome <input type="text" name="nome">
<p>
Inserisci il tuo cognome <input type="text" name="cognome">
<p>
<input type="submit" name="submit" value="invia">
</form>
```

Codice pagina html con uso metodo POST :

```
<form action="reg1.php" method="post">
Inserisci il tuo nome <input type="text" name="nome">
<p>Inserisci il tuo cognome <input type="text" name="cognome">
<p><input type="submit" name="submit" value="invia">
</form>
```

Possibile soluzione

Eseguendo in ambiente **EasyPHP – Sito locale** (file memorizzati in sottocartella ***www***)



Codice pagina PHP (in risposta ad invio dati con metodo **get**) con possibile integrazione html

```
<html>
<head>
<?php
                                        // recupero dati inviati
      $nome=$_GET['nome'];
      $cognome=$_GET['cognome'];
?>
</head>
<body>

<?php
       echo"<b>Ciao ".$nome." ".$cognome." </b>";
?>
</body>
</html>
```

Codice pagina PHP (in risposta ad invio dati con metodo **post**):

```
<?php
      $nome=$_POST['nome'];           // recupero dati inviati
      $cognome=$_POST['cognome'];
                                      // visualizzazione di stringhe concatenate
      echo"<b>Ciao ".$nome." ".$cognome." </b>";

?>
```

PHP e file testo (.txt)

PHP è un linguaggio di programmazione autonomo derivante dal C, ed è nato per soddisfare alcune esigenze che avevano i programmatori internet.

Questo vuol dire che il binomio PHP-MySql (o qualunque altro database) è nato dopo il linguaggio di programmazione.

PHP permette oltre l'utilizzo dei database, anche il lavoro con i file di testo (.txt). Mediante PHP possiamo ***leggerli, modificarli, crearli e salvarli***. In pratica possiamo avere un'alternativa ai database (anche se con molti limiti e molto più complicata).

Prima di analizzare le funzioni[3] relative vi informiamo che un eventuale errore con i file di testo genera un risultato booleano *false* con un errore mostrato a video.

Per ovviare a questo problema possiamo inserire il simbolo @ che non fa altro che nascondere l'errore generato. Per verificare l'errore possiamo associare un ciclo if al valore booleano.

Vediamo le **funzioni base** che consentono di lavorare con i file di testo:

fopen();

La funzione **fopen();** apre un file. Come per le connessioni con i database, solitamente si associa una variabile a questa connessione, la quale identifica il file in maniera univoca.

Esempio:

```
<?
$var=fopen("nome_file.txt","tipo");
?>
```

Tipo:

Al posto di *'tipo'* dobbiamo inserire una costante che identifica il tipo di apertura file.

R	Per sola lettura.
R+	Per lettura e scrittura (scrive all'inizio del file).
W	Per sola scrittura (I dati già scritti andranno persi e se il file non esiste sarà creato).
W+	Per scrittura e lettura (I dati già scritti andranno persi e se il file non esiste sarà creato. Scrive all'inizio del file).
A	Per aggiungere (I dati saranno aggiunti in coda a quelli già scritti, se il file non esiste sarà creato.)
A+	Per aggiungere e lettura (I dati saranno aggiunti in coda a quelli già scritti, se il file non esiste sarà creato.)

[3] Per un elenco di tutte la **funzioni del linguaggio**: http://docs.php.net/manual/it/funcref.php

fread();
La funzione **fread();** estrae un numero di byte dal file di testo (.txt).
Esempio:

```
<?
$var=fopen("nome_file.txt","r");
$var2=fread($var,20);
?>
```

In questo caso abbiamo aperto un file di testo (*nome_file.txt*) solo per leggerlo (*r*) ed abbiamo estratto 20 byte.
Per leggere tutto il file dobbiamo scrivere:

```
<?
$var=fopen("nome_file.txt","r");
$var2=fread($var,filesize("nome_file.txt"));
?>
```

Dove la funzione **filesize()** restituisce la grandezza totale del file.
Nel caso dentro il file ci siano dei rientri a capo che vogliamo visualizzare a video dobbiamo inserire un'altra funzione:
nl2br() che in pratica aggiunge un <br> quando trova un ritorno a capo nel file di testo.
Esempio:

```
<?
$var=fopen("nome_file.txt","r");
$leggi=fread($var,filesize("nome_file.txt"));
$sss=nl2br($leggi);
fclose($var);
echo $sss;
?>
```

fwrite();
La funzione **fwrite();** scrive una stringa nel file di testo che abbiamo aperto.
Esempio:

```
<?
$var=fopen("nome_file.txt","a+");
fwrite($var, "stringa di prova");
?>
```

In questo caso abbiamo aperto un file di testo (*nome_file.txt*) per aggiungere e lettura (*a+*) ed abbiamo scritto "*stringa di prova*" alla fine dei dati già scritti. Se il file non esisteva abbiamo creato un nuovo file di testo con sopra scritto "*stringa di prova*".

fclose();
La funzione **fclose();** chiude la connessione aperta verso il file di testo.
Esempio:

```
<?
$var=fopen("nome_file.txt","a+");
....
....
fclose($var);
?>
```

feof();
La funzione **feof();** verifica se abbiamo raggiunto la fine del file di testo che abbiamo letto.
Restituisce ***true*** se è andata a buon fine e ***false*** se viceversa abbiamo errato qualche passaggio.

file_exists();
Controlla se un file esiste. Risponde **true** se esiste e **false** se non esiste.
Esempio:

```
<? $var= file_exists("nome_file.txt");
if ($var==true) echo"true<br>";
if ($var==false) echo"false<br>";
?>
```

filesize();
Restituisce la grandezza totale del file che indichiamo all'interno delle parentesi.
Si può usare per scrivere un intero file, vedi l'esempio della funzione **fread();**

include();
Utilizzato anche per altri scopi può servire per scrivere a video un file di testo. In pratica include nella pagina web il testo contenuto nel file di testo. Che sia testo, comandi, uno script, qualunque sia il contenuto verrà inserito nella pagina web in quel preciso punto dove è inserito il comando *'include'*.

Esempio:

```
<? include("nome_file.txt"); ?>
```

Inserisce il testo senza tenere conto dei ritorni a capo, sempre che nel testo non ci siano dei tag **
** di HTML.

file();
La funzione **file("nome_file.txt")** restituisce un array con gli elementi uguali ad ogni riga del file di testo.
Possiamo capire quanto sia importante questa funzione. Unita alle altre funzioni array è possibile manipolare un file di testo quasi come un database.

count();
Questa non è una funzione direttamente collegabile ai file di testo, ma il suo inserimento in questa pagina è giustificato dal suo largo utilizzo con l'istruzione *file();*
L'istruzione **count();** conta gli elementi contenuti in un array.

Conclusioni:
I file di testo possono essere usati per molteplici usi. Dobbiamo solo immaginare il file di testo come un grosso array, ed operare di conseguenza.

- Immaginiamo di memorizzare i dati in un file di testo.
- In ogni riga un dato relativo ad un utente.
- A questo punto contiamo le righe (*file();* e *cont();*).
- Adesso con un ciclo for facciamo una scansione del contenuto array e troviamo quello cercato.

Appendice

Programmazione orientata agli oggetti in PHP

La possibilità di lavorare con classi e oggetti in PHP è stata introdotta con le seguenti versioni

- supporto di base agli oggetti da **PHP 4.0**
- supporto completo agli oggetti e migliori performance da **PHP 5.0**

In PHP le variabili e i metodi degli oggetti possono essere specificati a priori attraverso la definizione di classi

È anche possibile (ma sconsigliato) aggiungere nuove variabili d'istanza a oggetti già creati (in stile JavaScript)

Altri metodi per interagire con le basi di dati

- **L'estensione MySQL *improved* (MySQLi)**

  La libreria di funzioni per accedere a MySQL viste nella trattazione precedente fanno parte dell'***estensione PHP detta MySQL***

  Dalle versioni di MySQL 4.1.3 e successive, disponibile per PHP 5 e release superiori. è stata messa a disposizione di PHP una nuova ***estensione detta MySQLi*** (MySQL ***improved***) con un'interfaccia object oriented e alcune nuove funzionalità.

  Sebbene la vecchia estensione MySQL sia ritenuta al momento ***più stabile*** di MySQLi, quest'ultima diventerà nel prossimo futuro l'estensione "ufficiale"

  L'uso di MySQLi non è in realtà molto diverso dall'uso di MySQL.

- **L'estensione SQLite**

  MySQL è sempre stato il DBMS più comunemente usato con PHP
  Un'istallazione (o configurazione) minimale di PHP potrebbe però non comprendere MySQL
  Inoltre per piccole basi di dati si potrebbe preferire qualcosa di più semplice di MySQL

  Dalla versione PHP 5.0 è stata inclusa la nuova ***estensione SQLite***
  SQLite consiste di una **libreria** (sia procedurale che object-oriented) per memorizzare dati in piccoli database memorizzati come singoli file binari
  Trattandosi di una libreria e non di un DBMS, SQLite esiste solo all'interno dei linguaggi di programmazione che la implementano (tra cui PHP). Non si può accedere ai dati tramite console, o simili.

Alcuni aspetti di sicurezza in PHP[4]

L'utilizzo di PHP e MySQL espone ad alcuni possibili attacchi alla sicurezza del sistema
Anche nei piccoli siti web è bene prestare attenzione agli aspetti di sicurezza!
I tipi di attacchi più semplici da realizzare consistono nell'effettuare richieste all'applicazione PHP nascondendo frammenti di codice nei parametri:

- **MySQL Injection Attacks**: frammenti di codice SQL da far eseguire all'applicazione PHP
- **Cross-site Scripting (XSS) Attacks**: frammenti di codice HTML o JavaScript da memorizzare nella base di dati e successivamente eseguiti da altri client

[4] Interessante http://www.sicurezzaeinformatica.com/guida-sicurezza-php/