

E-book sul PHP



Teoria e molti esercizi

prof. Paolo Latella
Responsabile Laboratorio Mercurio
Istituto Tecnico Economico A. Bassi Lodi

Versione 1.0

Opera totalmente gratuita

Indice

- **Premessa**
- **Cos'è il PHP?**
- **Vantaggi di PHP**
- **Primo esempio**
- **PHP info**
- **HTTP Server Agent**
- **Variabili**
- **Assegnazione di valore a variabili**
- **Tipi di dato**
- **Esempi**
- **Assegnamento by value e by reference**
- **Tipi di base: stringhe**
 - Stringhe: single quoted
 - Stringhe: double quoted
 - Stringhe: Heredoc
 - Stringhe: lunghezza.
 - **Lezione completa sulle stringhe in PHP:**
 - ☞ Introduzione
 - ☞ Delimitare le stringhe
 - ☞ Visualizzare a schermo le stringhe
 - ☞ Funzioni per le stringhe in applicazioni web
 - ☞ Funzioni per formattare le stringhe
 - ☞ Stringhe e array
 - ☞ Conclusioni sulle stringhe
- ☞ **Gli Array (vettori):**
 - Introduzione
 - l'indice dei vettori
 - Gli array scalari e gli array associativi
- ☞ **Gli operatori aritmetici**
- ☞ **Gli operatori di confronto**
- ☞ **Gli operatori logici**
- ☞ **Strutture di controllo:**
 - Strutture di controllo fondamentali:
 - Sequenza
 - Goto (non è consigliabile l'utilizzo)
 - Strutture di controllo della programmazione strutturata
 - Alternativa

- Alternativa if-then e if-then-else
- L'alternativa case
- Iterazione:
- Ciclo for
- Ciclo while
- Ciclo loop-until
- Varianti di while e loop-until
- Iterazione basata su collezioni
- Terminazione anticipata di cicli e iterazioni

- ▤ **if (in Php)**
- ▤ **endif if (in Php)**
- ▤ **if-else if (in Php)**
- ▤ **if-elseif-else if (in Php)**
- ▤ **while if (in Php)**
- ▤ **for if (in Php)**
- ▤ **Foreach if (in Php)**
- ▤ **switch-case-default if (in Php)**

■ **Form HTML: sintassi**

■ **Form HTML**

■ **Accedere alle variabili di una FORM HTML**

■ **Esempio**

■ **Inviare una mail in PHP e gestire le form:**

- Prerequisiti
- Interagiamo con i moduli (form)
- la funzione include()

■ **FormMail Multiplo in php: come comporre e gestire i vari campi di un modulo**

■ **I file di testo:**

- Introduzione
- Le operazioni più comuni, universalmente presenti in tutti i sistemi operativi:


- Apertura
- Lettura
- Scrittura
- Chiusura

■ **Operazioni del sistema operativo sui file:**

- Riallocazione
- Rinominazione
- Eliminazione


▤ **Apertura di un file di testo in Php**

▤ **Lettura da file in php**

 **Scrittura su file in php**

 **Scrittura su file – 2 in php**

 **La gestione completa del file di testo usando i vettori per la modifica e la cancellazione senza usare un secondo file**

 **L'aggiornamento e la cancellazione di un record da un file di testo usando due archivi txt (metodo sicuro senza il rischio di perdere i dati). Ecco le tre pagine in Php (inserimento, modifica e cancellazione)**

Premessa

Questo E-book contiene la parte teorica e pratica del linguaggio di programmazione PHP spiegato e “provato” in laboratorio dai miei alunni in questo ultimo anno di scuola 2012-2013. E' stato realizzato prendendo spunto da alcuni siti esterni e dal mio blog didattico <http://paololatella.blogspot.it> . Nel blog didattico è possibile approfondire diversi argomenti che si trovano in questo libro elettronico.

Perché questo E-book? Semplicemente per due motivi, offrire un prodotto editoriale totalmente gratuito agli studenti dell'Istituto Tecnico Economico A. Bassi di Lodi indirizzo Sistemi Informativi Aziendali (programmatori) e nello stesso tempo rispondere alla traccia dell'esame che sto preparando al DOL del Politecnico di Milano, dove si richiede appunto la realizzazione di un E-book didattico.

Ho scelto di aggiornarmi seguendo il DOL il corso biennale del Politecnico di Milano sulle nuove tecnologie rivolto a tutti i docenti di ruolo di ogni ordine e grado per certificare le mie competenze acquisite in 28 anni di insegnamento.

DOL (Diploma On Line) è un programma dedicato alla formazione di insegnanti esperti nell'uso delle tecnologie nella didattica.

Relativamente al E-book, essendo la prima versione, ci saranno probabilmente errori di battitura che verranno corretti nei prossimi mesi.

In questo ipertesto ci sono concetti sul PHP, esercizi svolti dalle classi terze e quarte, dal semplice esercizio di una somma tra due variabili fino alla gestione dei file di testo con il Php. Ci sono inoltre le guide per installare i server web sia su notebook con sistema operativo Windows che Apple Mac.

Un ringraziamento a tutti coloro che hanno realizzato i documenti pubblici e pubblicati sul Web dai quali ho tratto questa dispensa per i miei alunni, totalmente gratuita e senza scopo di lucro alcuno e sono:

- 📖 Dispense per il corso di Linguaggi e Traduttori - Facoltà di Economia - Università di Trento - prof. Paolo Bouquet
- 📖 Il blog didattico del prof. Paolo Latella <http://paololatella.blogspot.it>
- 📖 <http://www.mrwebmaster.it>
- 📖 XAMPP per Mac OS X di Kristian Marcroft
- 📖 <http://www.php.net/manual/en/ref.strings.php>
- 📖 <http://vademecum.aruba.it>
- 📖 Il capitolo sul modulo form a cura di Claudio Curci - Aggiornamenti di Max Kiusso
- 📖 <http://www.web-link.it/php/index7.php>
- 📖 http://www.miniscript.it/articoli/57/la_gestione_di_database_testuali_txt.html
- 📖 <http://it.wikipedia.org/wiki/Array>
- 📖 Foto copertina: <http://www.talentsfromindia.com/wp-content/uploads/2010/12/php-development.jpg>

Buona lettura e naturalmente buon studio!

Per maggiori informazioni e chiarimenti potete contattarmi all'indirizzo paolo.latella@alice.it.

Cos'è il PHP?

E' un linguaggio di *scripting server side*

La differenza tra lato client e lato server sta tutta nel modo e da chi viene interpretata una pagina Web quando essa viene caricata.

quando un server Web predisposto per il PHP riceve una richiesta dal browser di un client iniziano una serie di operazioni: Il server:

Legge ed individua la pagina sul server.

Esegue le istruzioni PHP contenute all'interno della pagina ed esegue tutti i comandi PHP.

Rimanda la pagina risultante al Browser.

Vantaggi di PHP

PHP è un linguaggio molto semplice da utilizzare, a cominciare dalla sintassi derivata direttamente da veri linguaggi di programmazione come C/C++, Perl, Java.

Forse la vera forza del PHP sta nella gestione dei database, con poche righe di codice è possibile accedere qualsiasi database, estrapolare i dati che ci interessano e inserirli nella pagina Web.

Un altro punto a favore del PHP è la sua natura OpenSource, quindi gratuita.

Infine il PHP gira su tutti in principali Web server ed in linea di massima non dobbiamo apportare nessuna modifica al codice quando lo spostiamo da un Web server ad un altro.

Primo esempio

Affinché l'interprete PHP riesca a distinguere all'interno del codice il linguaggio da interpretare ed eseguire (PHP) dall'HTML occorre utilizzare dei TAG particolari. Ecco un semplice esempio:

```
<html>
<body>
  <h1>
    <?php echo "Hello World!!"; ?>
  </h1>
</body>
</html>
```

Risultato

Al client arriverà il seguente file html:

```
<html>
<body>
  <h1>Hello World!! </h1>
</body>
</html>
```

PHP info

Un altro esempio è il comando che ci dà informazioni sulla configurazione del server. Se dentro al file test.php si mette la seguente linea:

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

si ottengono tutte le informazioni sulla configurazione del web server e del php su cui si sta lavorando.

HTTP Server Agent

Lo script:

```
<html>
```

```
<body>
```

Il server agent che stai usando `;:

```
<center>
```

```
<?php echo $_SERVER["HTTP_USER_AGENT"]; ?>
```

```
</center>
```

```
</body>
```

```
</html>
```

HTTP Server Agent

.... produrrà (sul mio PC) il seguente output:

```
<html>
```

```
<body>
```

Il server agent che stai usando `;:

```
<center>
```

Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)

```
</center>
```

```
</body>
```

```
</html>
```


Variabili

Le variabili sono come dei “cassetti” in cui si possono mettere dei valori.

Ogni variabile ha dunque un nome e un valore.

Il nome di una variabile in PHP:

è sempre prefissato dal simbolo del dollaro (\$)

contiene caratteri alfanumerici e inizia con una lettera o con il simbolo “_”

In PHP, una variabile è creata automaticamente ogni volta che le si assegna un valore (dichiarazione implicita)

Assegnazione di valore a variabili

In PHP, una variabile è creata automaticamente ogni volta che le si assegna un valore (dichiarazione implicita)

Per esempio, con il comando:

```
$a = 5;
```

Facciamo due cose contemporaneamente:

Creiamo una nuova variabile con nome \$a

Le assegniamo il valore “5”

Tipi di dato

I tipi di dato supportati da PHP si distinguono in:

tipi scalari:

numeri (interi e a virgola mobile)

stringhe

booleani

tipi composti:

array

oggetti

Esempi

Assegnamento di variabili:

numeri: `$a = 5; $p_greco = 3.14;`

stringhe: `$a = "Hello world!";`

booleani: `$t = TRUE; $f = FALSE;`

array (esplicito):

`$giorni = array("lun", "mar", "mer", "gio", "ven", "sab", "dom");`

array (implicito):

`$giorni[0] = "lun";`

`$giorni[1] = "mar";`

...

`$giorni[6] = "dom";`

Assegnamento

by value e by reference

E' possibile assegnare il valore di una variabile a un'altra variabile in due modi diversi:

by value: "copiando" nella seconda variabile il valore della prima (vedi esempio 9-1):

`$a = 10;`

`$b = $a;`

by reference: creando un "link" tra il valore della seconda variabile e il valore della prima (vedi esempio 9-2)

`$a = 10;`

`$b = &$a;`

Tipi di base: stringhe

Una stringa può essere specificata in 3 modi:

Single quoted (')

Double quoted (")

Heredoc syntax

Stringhe: single quoted

Le stringhe possono essere introdotte tra singoli apici:

```
<?php echo 'Stringa'; ?>
```

Le stringhe così introdotte possono estendersi su più righe:

```
<?php echo 'Si possono usare la forma single quoted per definire stringhe su più righe'; ?>
```

Si può usare la controbarra (\) per utilizzare simboli speciali:

```
<?php echo 'Mario disse: "Torno all\'una";  
echo 'Hai cancellato C:\*. *?'; ?>
```

Il singolo apice non permette di introdurre nuove righe:

```
<?php echo 'Questo non produce: \n una nuova riga'; ?>
```

All'interno di singolo apice le variabili non vengono espanse:

```
<?php $variabili = 23456;  
echo 'Neanche le $variabili vengono espanse'; ?>
```

Stringhe: double quoted

Le stringhe introdotte tra doppi apici permettono di:

Utilizzare più simboli di escape:

\n : nuova riga

\t: tabulazione

\r: return

\": carattere doppio apice

Espandere le variabili:

```
<?php $a = 10;  
echo "Il valore di $a è $a"; ?>
```

Concatenare stringhe:

```
<?php $a = "Universita"; $b = "di"; $c = "Trento";  
echo "$a" . "$b" . "$c"; ?>
```

Stringhe: Heredoc

Esiste anche una terza sintassi per introdurre stringhe, detta Heredoc:

```
<?php
    $a = <<<EOD
    Ecco una stringa
    su più righe
    con sintassi heredoc
    EOD;
?>
```

Stringhe: lunghezza

La funzione strlen permette di calcolare la lunghezza di una stringa:

```
$a = "prova";
echo 'strlen("prova") = ' . strlen($a);
```

L'output sarà:

```
strlen("prova") = 5
```

Lezione sulle stringhe in PHP

Introduzione

Con PHP risulta molto semplice manipolare le stringhe, in parte per le caratteristiche del linguaggio dotato, come è noto, di una sintassi semplice, snella e compatta. In parte per la disponibilità di una vasta libreria di funzioni (consultabile alla url <http://www.php.net/manual/en/ref.strings.php>) che permette di svolgere con facilità le più comuni come le più complesse elaborazioni.

In questo articolo si introdurranno i concetti basilari relativi alle stringhe, per poi passare ad illustrare alcune tra le numerose funzioni, soprattutto nell'ottica dello sviluppo di applicazioni web. Gli argomenti trattati si rivolgono principalmente a chi da poco si è avvicinato a PHP, malgrado ciò qualche spunto interessante potranno trovarlo anche i programmatori un po' più esperti. L'obiettivo è fornire una breve panoramica degli strumenti messi a disposizione dal linguaggio per scrivere codice efficiente ed elegante senza "reinventare l'acqua calda".

Delimitare le stringhe

Con il termine stringa si intende una sequenza di caratteri: "web", "consulta freephp.html.it", "PHP rende la vita facile ai programmatori", sono semplici esempi di stringhe. Per assegnare ad una variabile un valore di tipo stringa dobbiamo utilizzare dei delimitatori che consentano di racchiudere la sequenza di caratteri desiderata. La sintassi del linguaggio fornisce varie alternative, ciascuna con proprie particolarità.

Come primo approccio possiamo delimitare la stringa mediante **virgolette doppie** ("):

```
<?php
$stringa = "Naviga su html.it";
// posso anche andare a capo
$stringa1 = "Naviga
su html.it
lo troverai interessante";
?>
```

Quando si adotta questa sintassi PHP analizza il contenuto della stringa alla ricerca di nomi di variabile individuati dal simbolo \$; qualora ne trovi uno lo sostituisce automaticamente con il corrispondente valore. Questa caratteristica prende il nome di espansione dei nomi di variabile. Il codice seguente esemplifica quanto descritto:

```
<?php
$sito = "html.it";
// la variabile $stringa conterrà
// la stringa "Naviga su html.it"
$stringa = "Naviga su $sito";
```

```
// posso utilizzare anche la componente di un array
$arraysito[0] = "html.it";
$stringa1 = "Naviga su $arraysito[0]";
?>
```

Si osservi che l'automatismo della sostituzione necessita di alcuni accorgimenti se la variabile all'interno della stringa è un array associativo o bidimensionale:

```
<?php
$sito["nome"] = "html.it";
// la successiva assegnazione genererà un errore
$stringa1 = "Naviga su $sito["nome"]";
// anche questa assegnazione genererà un errore
$stringa2 = "Naviga su $sito['nome']";
// questa assegnazione funziona correttamente
$stringa3 = "Naviga su $sito[nome]";
// si può ricorrere alle parentesi graffe
// anche questa assegnazione funziona
$stringa4 = "Naviga su {$sito['nome']}";
// anche la variabile $stringa5 conterrà la stringa
// corretta "Naviga su html.it "
$stringa5 = "Naviga su " . $sito["nome"];
$sito[4][1] = "html.it";
// la variabile $stringa6 conterrà la stringa
// errata "Naviga su Array[1]"
$stringa6 = "Naviga su $sito[4][1]";
// la successiva variabile conterrà la stringa
// corretta "Naviga su html.it "
$stringa7 = "Naviga su " . $sito[4][1];
// anche la successiva variabile conterrà la stringa corretta
$stringa8 = "Naviga su {$sito[4][1]}";
?>
```

Come si può notare nel caso della variabile \$stringa3, la componente di un array associativo all'interno di una stringa non va specificata mediante virgolette doppie o singole. Questo modo di procedere si contrappone alle usuali regole di buona scrittura del codice indicate nella documentazione ufficiale. In alternativa si può ricorrere all'utilizzo delle parentesi graffe ovvero alla cosiddetta "sintassi complessa", come nel caso delle variabili \$stringa4 e \$stringa8 dell'esempio precedente. Il nome deriva non tanto dalla sua difficoltà di utilizzo, quanto dalla possibilità di inserire espressioni complesse come se si fosse all'esterno della stringa, per maggiori dettagli si rimanda alla documentazione ufficiale.

Infine nel caso delle variabili \$stringa5 e \$stringa7 si è fatto ricorso all'operatore di concatenazione, ovvero il punto (.), che permette di unire più stringhe. Il successivo esempio ne rende più chiaro l'utilizzo:

```
<?php
$uno = "Naviga";
$due = "su";
$tre = "html.it";
// la variabile $stringa conterrà la
// stringa "Naviga su html.it "
$stringa = $uno. " " . $due. " " . $tre;
?>
```

Utilizzando l'operatore .= è possibile effettuare concatenazione ed assegnazione in un'unica istruzione.

```
<?php
$stringa = "Naviga su ";
//sintassi classica
$stringa = $stringa. "html.it";
//sintassi abbreviata
$stringa .= "html.it";
?>
```

Se all'interno della stringa vogliamo inserire le virgolette doppie ("), il carattere di backslash (\), o il simbolo del dollaro (\$) dobbiamo effettuarne il cosiddetto **escape**. Sarà necessario, cioè, far precedere tali simboli dalla backslash così: \", \\, \\$, in caso contrario potremmo ottenere un messaggio d'errore. È anche possibile inserire alcuni caratteri speciali quali linefeed (new line), \n, carriage return, \r, o tab, \t.

```
<?php
$stringa = "Egli disse:\"Naviga su html.it!\\"";
$stringa1 = "Adesso vado a capo.\nQui sono alla riga successiva.";
?>
```

In alternativa PHP prevede che le stringhe possano essere racchiuse tra **virgolette singole** ('). Con questa sintassi non verrà effettuata la sostituzione delle variabili con il corrispondente valore.

```
<?php
$sito = 'html.it';
// la successiva variabile conterrà la stringa
// errata "Naviga su $sito"
$stringa1 = 'Naviga su $sito';
// la successiva variabile conterrà la stringa
// corretta "Naviga su html.it"
$stringa2 = 'Naviga su ' . $sito;
?>
```

I caratteri che in questo caso devono essere preceduti da backslash sono le virgolette singole, \' e la backslash stessa, \\.

```
<?php
// l'assegnazione seguente è corretta
$stringa1 = 'Egli disse:"L\'importante è navigare su html.it"';
// non è possibile usare caratteri speciali
$stringa2 = 'Il carattere di new line \n non verrà considerato, ma stampato tale e quale';
?>
```

Visualizzare a schermo le stringhe

Dopo aver illustrato come rappresentare le stringhe, vediamo ora come visualizzarle. PHP prevede diverse funzioni per l'output, la principale è **echo()**. In realtà non è una funzione vera e propria, ma un costrutto del linguaggio e quindi possiamo omettere le parentesi tonde.

```
<?php
```

```
echo "Consulta il sito html.it";
```

```
// la stringa può essere suddivisa su più linee
```

```
echo "La stringa si compone
```

```
di più linee
```

```
ma verrà correttamente stampata";
```

```
// si possono inserire dei caratteri new line \n all'interno della stringa
```

```
echo "La stringa contiene caratteri new line\nne viene stampata\nsu più linee\ncontrollate il sorgente html";
```

```
echo "La stringhe ", "possono essere più di una ";
```

```
$stringa = "stringa di prova";
```

```
echo $stringa;
```

```
echo "Ecco una $stringa";
```

```
?>
```

Echo prevede una sintassi abbreviata che funziona solo se nel file di configurazione php.ini la direttiva **short_open_tag** è impostata su on:

```
<!-- così viene stampata la variabile $stringa -->
```

```
<?=$stringa?>
```

La funzione **print()**, anch'essa un costrutto del linguaggio, si può considerare praticamente equivalente ad echo. Alcuni test hanno però dimostrato che echo risulta più veloce del 5-10% a seconda dei contesti. Si osservi che la sintassi di print() non permette argomenti multipli.

```
<?php
```

```
$stringa = "stringa di prova";
```

```
print "Ecco una $stringa";
```

```
// così non funziona
```

```
print "La stringhe ", "non possono essere più di una ";
```

```
?>
```

La funzione **printf()** (come l'omologa **sprintf()**), viene utilizzata per produrre un output formattato. È sconsigliabile in termini di prestazioni rispetto alle altre due, va quindi utilizzata solo quando sia effettivamente necessaria. Si pensi di dover stampare il numero 1/6, utilizzando echo o print si otterrebbe il risultato seguente 0.166666666667. Per ottenere un risultato più leggibile, con due sole cifre dopo la virgola dovremo ricorrere a printf();

```
<?php
```

```
// stampiamo (1/6) come 0.17
```

```
printf("%1.2f", (1/6));
```

```
?>
```

In pratica il primo argomento della funzione specifica come rappresentare il numero che compare come secondo argomento. Nell'esempio mostrato il numero da stampare dovrà

essere rappresentato mediante minimo un carattere, dovrà avere due cifre dopo la virgola e dovrà essere trattato come un numero floating-point (f).

L'utilizzo della funzione non è immediato, sinteticamente per indicare come formattare l'output si specificano nell'ordine:

- ☞ uno o più caratteri (escluso %) che precedono il risultato
- ☞ il simbolo %
- ☞ uno specificatore di padding (opzionale)
- ☞ uno specificatore di allineamento (opzionale)
- ☞ uno specificatore di numero minimo di caratteri (opzionale)
- ☞ uno specificatore di precisione (opzionale)
- ☞ uno specificatore di tipo che indica se l'argomento da stampare sia da considerarsi intero, float, stringa etc.

Per maggiori dettagli si rimanda alla documentazione ufficiale, mentre di seguito si riportano alcuni esempi significativi:

```
<?php
// stampiamo "valore pari a euro 30.25"
printf("valore pari a euro %2.2f", 30.25);
// stampiamo 2 come 00002
printf("%05d", 2);
// stampiamo 2 come ***2
printf("%'*4d", 2);
?>
```

PHP fornisce anche altre funzioni di output che, per ragioni di spazio e particolarità di utilizzo, non prenderemo in considerazione. Si descriverà più avanti la funzione `number_format()`, utile per la formattazione dei numeri, che permette di evitare il ricorso alla maggiormente complessa `printf()`.

Funzioni per le stringhe in applicazioni web

Terminata l'analisi dei concetti basilari, prendiamo in considerazione alcune funzioni particolarmente utili nello sviluppo di applicazioni web. Se siete arrivati fin qui, probabilmente questo è il vostro obiettivo. La regola fondamentale per chi si cimenti in questo compito è: non fidarsi mai di ciò che proviene dal web! In particolare dobbiamo trattare come "contaminati" tutti dati inviati dai navigatori del nostro sito, assicurandoci che il loro contenuto non possa in qualche modo danneggiare l'applicazione. L'inserimento di tag HTML, se inviati al browser, potrebbe rovinare il layout delle nostre pagine, l'inserimento di javascript potrebbe risultare ancora più dannoso. Basti pensare ad applicazioni quali forum o guestbook per rendersi conto dei possibili rischi. Fortunatamente viene in nostro aiuto la funzione **strip_tags()** che permette di eliminare i tag HTML, JavaScript e marcatori PHP indesiderati.

```
<?php
$stringa = "Visita <b><a href=\"http://www.html.it\">html.it</a></b>";
// stampiamo solo la stringa "Visita html.it"
// eliminando i tag html
echo strip_tags($stringa);
?>
```

La funzione è abbastanza flessibile da permettere di conservare solo i tag desiderati, basterà specificarli come secondo argomento:

```
<?php
$stringa = "Visita <b><i><a href=\"http://www.html.it\">html.it</a></i></b>";
// stampiamo la stringa "<b><i>Visita html.it</i></b>"
// eliminando solo i tag <a>
echo strip_tags($stringa,"<b><i>");
?>
```

La funzione **trim()** permette di rimuovere spazi bianchi, " " (ASCII 32), caratteri di new line, "\n" (ASCII 10), caratteri di carriage return, "\r" (ASCII 13), tabulazioni, "\t" (ASCII 9), NUL "\0" (ASCII 0), tab verticali, "\x0B" (ASCII 11) all'inizio ed alla fine di una stringa. Tali caratteri possono essere erroneamente introdotti in un campo di input o una textarea. Una situazione tipica si ha quando l'utente effettua un copia-incolla da un documento. Mediante trim() un simile dato può essere "ripulito" prima di essere elaborato.

Analogamente le due funzioni **ltrim()** ed **rtrim()** si limitano a rimuovere tali caratteri rispettivamente all'inizio ed alla fine della stringa.

```
<?php
$stringa = " ci sono degli spazi ";
// la variabile $stringa dopo l'assegnazione successiva conterrà la stringa
// "ci sono degli spazi" priva di spazi all'inizio e alla fine
$stringa = trim($stringa);
?>
```

Per visualizzare codice HTML, senza che questo venga interpretato dal browser, possiamo ricorrere alla funzione **htmlspecialchars()**. Nel suo utilizzo basilare si occupa di trasformare i caratteri <, >, ", & nei corrispondenti caratteri HTML, <, >, ", &. Si può anche optare per la più completa funzione **htmlentities()** che trasforma tutti i caratteri che hanno un equivalente HTML nella corrispondente entità HTML. Ad esempio il simbolo di marchio registrato ® viene convertito nel corrispondente ®.

```
<?php
$stringa = "<caratteri html>";
// la variabile $stringa dopo l'assegnazione successiva conterrà
// la stringa "&lt;caratteri html&gt;"
$stringa = htmlspecialchars($stringa);
echo $stringa;
?>
```

Nel lavorare con i database non possiamo trascurare la funzione **addslashes()**. Tale funzione effettua automaticamente l'escape dei caratteri che potrebbero generare errori nelle query SQL, inserendo opportunamente delle backslash. Si osservi il codice seguente:

```
<?php
// dati per la connessione al database
$server = "localhost";
$user = "utente";
$password = "xyz";
$database = "miodb";
$conn = @mysql_connect($server,$user,$password);
if ($conn)
{
    $stringa = "Ci vuole pratica per usare l'HTML";
    // l'apice genererà un errore nell'esecuzione della query seguente
    $query = "INSERT INTO nome_tabella (campo_stringa) VALUES ('$stringa')";
    $result = mysql_db_query($database,$query,$conn);

    // in questo secondo caso si evita l'errore trasformando la stringa
    // così: "Ci vuole pratica per usare l\HTML"
    $stringa = addslashes($stringa);
    $query = "INSERT INTO nome_tabella (campo_stringa) VALUES ('$stringa')";
    $result = mysql_db_query($database,$query,$conn);
}
?>
```

Nel primo caso SQL viene confuso dall'apostrofo (') presente nella stringa che può essere interpretato come delimitatore finale di campo generando un errore. Nel secondo, l'escape del carattere, fa sì che tutto funzioni correttamente.

Si osservi che se nel file php.ini la direttiva **magic_quotes_gpc** è impostata ad on, i dati provenienti da GET, POST e Cookie vengono modificati automaticamente senza bisogno di usare addslashes. Per ulteriori approfondimenti si rimanda alla documentazione ufficiale. L'operazione inversa viene svolta dalla funzione **stripslashes()** che rimuove tutte le backslash e restituisce la stringa originale. Ad esempio \' diventa ', \\ diventa \ e così via.

Funzioni per formattare le stringhe

Passiamo ora ad analizzare il tema più complesso della formattazione avanzata.

Spesso nella stampa di una stringa vogliamo ottenere una particolare formattazione, anche in questo caso PHP ci favorisce mettendoci a disposizione un buon numero di funzioni.

Vediamone alcune di frequente utilizzo: **strtoupper()** e **strtolower()** permettono di convertire, rispettivamente in maiuscolo e minuscolo, i caratteri alfabetici di una stringa.

```
<?php
$stringa = "Lavoriamo sulle lettere Maiuscole e Minuscole";
// stampiamo "LAVORIAMO SULLE LETTERE MAIUSCOLE E MINUSCOLE"
echo strtoupper($stringa);
// stampiamo "lavoriamo sulle lettere maiuscole e minuscole"
echo strtolower($stringa);
?>
```

La funzione **ucfirst()** rende maiuscolo il carattere iniziale di una stringa, mentre **ucwords()** rende maiuscolo il primo carattere di ciascuna parola in essa contenuta.

```
<?php
$stringa = "consulta il sito html.it";
// stampiamo "Consulta il sito html.it"
echo ucfirst($stringa);
// stampiamo "Consulta Il Sito Html.it"
echo ucwords($stringa);
?>
```

Una funzione che spesso risulta utile è **nl2br()**, essa consente di trasformare i caratteri di new line (\n) nei corrispondenti
 HTML. Un possibile impiego lo si ha nella stampa di testo inviato mediante una textarea (magari memorizzato in database), in modo da mantenere i ritorni a capo originari.

```
<?php
$stringa = "uno
due
tre
";
echo nl2br($stringa);
/* restituisce
uno<br>
due<br>
tre<br>
*/
?>
```

La funzione **wordwrap()** suddivide la stringa passata come primo argomento, in base al numero di caratteri specificati come secondo argomento. L'eventuale terzo argomento rappresenta il carattere scelto come delimitatore, di default viene usato \n. Se la lunghezza non viene specificata la stringa verrà interrotta a 75 caratteri. L'ultimo argomento, sempre opzionale, se impostato a 1 impone che la stringa sia suddivisa alla lunghezza prevista anche se ciò spezza in più parti una singola parola. Vediamo gli esempi seguenti:

```
<?php
$testo = "Cantami o diva del pelide Achille l'ira funesta";
```

```
// spezziamo il testo ogni 15 caratteri
$testo1 = wordwrap($testo,15);
// trasformo i \n in <br> per la stampa html
echo nl2br($testo1);
/* restituisce
Cantami o diva
del pelide
Achille l'ira
funesta
*/
// spezziamo il testo ogni 6 caratteri
// senza mantenere intere le parole
echo nl2br(wordwrap($testo,6,"\n",1));
/* restituisce
Cantam
i o
diva
del
pelide
Achill
e
l'ira
funest
a
*/
?>
```

Anche se non riguarda strettamente le stringhe, riprendiamo la già citata funzione **number_format()** utilizzabile per formattare i numeri evitando il ricorso alla più complessa `printf()`. Il primo argomento sarà il numero da formattare, il secondo il numero di cifre dopo la virgola, con il terzo ed il quarto si possono opzionalmente specificare i separatori delle cifre decimali e delle migliaia.

```
<?php
// stampiamo 1/6 come 0.17 anziché 0.166666666667
echo number_format((1/6),2);
// se specificato il secondo argomento viene usata la virgola
// come separatore delle migliaia: 10000 diventa 10,000.00
echo number_format(10000,2);
// le cifre decimali saranno separate da v e
// le migliaia da p 3567,90 diventa 3p567v90
echo number_format(3567.90,2,"v","p");
?>
```

Vediamo ora alcune funzioni che svolgono operazioni comuni nel lavorare con le stringhe. La funzione **strlen()** fornisce il numero di caratteri di cui si compone una stringa:

```
<?php
// la funzione stamperà il valore 14
echo strlen("Visita html.it");
?>
```

Per estrarre una porzione da una stringa o, come si suol dire, per ottenere una sottostringa, possiamo ricorrere a **substr()**. Questa funzione prevede tre argomenti: la stringa da elaborare, il punto di partenza a partire dal quale estrarre la sottostringa ed eventualmente (parametro opzionale) la lunghezza della sottostringa. A tale proposito si rammenta che nel computo il primo carattere ha posizione 0 e non 1. Gli esempi seguenti chiariranno quanto descritto:

```
<?php
// stamperà "html.it" la porzione
// dalla posizione 7 alla fine
echo substr("Visita html.it",7);
// stamperà "html" partendo dalla
// posizione 7 per 4 caratteri di lunghezza
echo substr("Visita html.it",7,4);
?>
```

Se come punto di partenza si specifica un valore negativo, si partirà a ritroso dalla fine della stringa:

```
<?php
// stamperà "t"
echo substr("Visita html.it",-1);
// stamperà "it"
echo substr("Visita html.it",-2);
// stamperà "html"
echo substr("Visita html.it",-7,4);
?>
```

Se si inserisce un valore negativo per la lunghezza, verranno omessi tanti caratteri dalla fine della stringa quanti specificati dal valore:

```
<?php
// stamperà "Visita" eliminando
// 8 caratteri a partire dalla fine della stringa
echo substr("Visita html.it",0,-8);
?>
```

Spesso utili in combinazione con le precedenti funzioni sono **strpos()** e **strrpos()**. Con **strpos()** è possibile individuare la posizione della prima occorrenza di una stringa all'interno di un'altra. Il primo argomento indicherà la stringa all'interno della quale effettuare la ricerca, il secondo specificherà invece la stringa che si vuole cercare. Si faccia attenzione che la funzione è case sensitive, ovvero distingue maiuscole e minuscole, inoltre, come visto in precedenza, il primo carattere ha posizione zero.

```
<?php
// la funzione stamperà la posizione della prima i ovvero 1
// si ricordi che il primo carattere ha posizione 0
echo strpos("Visita html.it","i");
// la funzione stamperà 7
echo strpos("Visita html.it","html");
?>
```

La funzione prevede un terzo argomento opzionale per specificare da che posizione iniziare la ricerca:

```
<?php
// stamperà la posizione della prima i dopo
```

```
// il terzo carattere ovvero 3
echo strpos("Visita html.it","i",2);
?>
```

Analogamente la funzione `strrpos()` fornirà la posizione dell'ultima occorrenza del carattere specificato. Se come secondo argomento viene inserita una stringa verrà considerato solo il primo carattere:

```
<?php
// la funzione stamperà la posizione dell'ultima i ovvero 12
echo strrpos("Visita html.it","i");
// la funzione stamperà ancora 12
echo strrpos("Visita html.it","ixxxxxxxxxxxx");
?>
```

Un possibile problema si può verificare se la stringa non viene trovata, infatti entrambe le funzioni restituiscono il valore booleano `FALSE`, che, semplificando, può considerarsi equivalente a zero, ovvero alla posizione del primo carattere. Per evitare errori basta utilizzare l'operatore `===` per verificare il valore fornito dalla funzione.

```
<?php
$posizione = strpos("Visita html.it","V");
// questo codice induce in errore
if(!$posizione)
{
    // stringa non trovata
}
else
{
    // stringa trovata
}
// questo codice è corretto
if($posizione === false)
{
    // stringa non trovata
}
else
{
    // stringa trovata
}
?>
```

Per effettuare sostituzioni di caratteri all'interno di una stringa si può ricorrere a **`strtr()`**, tale funzione riceve come primo argomento la stringa da elaborare, come secondo i caratteri da sostituire, come terzo i caratteri utilizzati per la sostituzione.

```
<?php
// la funzione sostituirà ogni a con o ed ogni i con u
// si otterrà "Vusuto html.ut";
echo strtr("Visita html.it","ai","ou");
?>
```

Sempre in tema di sostituzioni PHP fornisce la funzione **`str_replace()`** che sostituisce tutte le occorrenze del primo argomento con la stringa specificata come secondo argomento, agendo sulla stringa indicata come terzo argomento.

```
<?php
// la funzione sostituirà "Ermanno" con
// "Ermanno Ancona"
$stringa = "Articolo a cura di Ermanno";
echo str_replace("Ermanno","Ermanno Ancona",$stringa);
?>
```

Si osservi che per realizzare regole di sostituzione più complesse, basate su espressioni regolari, si può ricorrere alle funzioni `ereg_replace()` o `preg_replace()`, non discusse in questo articolo.

Stringhe e array

Vi è una certa analogia tra stringhe ed array, addirittura potremmo considerare una stringa come un array di caratteri. Il primo carattere rappresenta la componente 0, il secondo la componente 1 e così via.

```
<?php
$stringa = "Visita il sito html.it";
// verrà stampato il carattere "V"
echo $stringa[0];
// verrà stampato il carattere "t"
echo $stringa[4];
?>
```

La sintassi mostrata nell'esempio precedente viene mantenuta per compatibilità con le versioni precedenti, ma è sconsigliata dalla versione 4 di PHP. Viene invece preferito l'utilizzo delle parentesi graffe, vediamo alcuni esempi:

```
<?php
$stringa = "Visita il sito html.it";
// verrà stampato il carattere "V"
echo $stringa{0};
// stampiamo l'ultimo carattere della stringa "t"
// usando la sintassi complessa
echo $stringa{strlen($stringa)-1};
// sostituiamo l'ultimo carattere della stringa
// usando la sintassi complessa
$stringa{strlen($stringa)-1} = 'k';
?>
```

Alcune funzioni permettono la conversione di stringhe in array e viceversa. La funzione **explode()** riceve come primo argomento il o i caratteri che separano i diversi elementi, come secondo argomento la stringa che si vuole convertire in array. Come al solito gli esempi chiariranno il concetto:

```
<?php
$stringa = "12,2,6,78,89,56";
// convertiamo in array utilizzando la virgola come separatore
$array = explode(",", $stringa);
echo $array[0]; // stampa 12
echo $array[1]; // stampa 2
echo $array[2]; // stampa 6
.....
?>
```

Si può, opzionalmente, utilizzare un terzo argomento per limitare il numero di componenti dell'array:

```
<?php
$stringa = "12,2,6,78,89,56";
// convertiamo in array di tre componenti
$array = explode(",", $stringa, 3);
echo $array[0]; // stampa 12
echo $array[1]; // stampa 2
```

```
echo $array[2]; // stampa 6,78,89,56
?>
```

Si noti che se il/i caratteri indicati come separatore non si trovano all'interno della stringa, verrà creato un array di una sola componente contenente tutta la stringa.

Per ottenere l'effetto opposto, ovvero convertire un array in una stringa basta ricorrere alla funzione **implode()**. Il primo argomento sarà la stringa utilizzata come separatore, il secondo l'array da convertire.

```
<?php
$array[0] = "visita"; $array[1] = "il"; $array[2] = "sito";
// convertiamo in stringa usando la virgola come separatore
// il risultato sarà "visita,il,sito";
$stringa = implode(",",$array);
echo $stringa;
?>
```

Merita un breve cenno la funzione **split()**, più sofisticata di **explode** anche se più lenta. Essa infatti permette di specificare un'espressione regolare come separatore. L'esempio che segue indica come separatore un tabulatore o una nuova linea:

```
<?php
$stringa = "Visita    il    sito
html.it
molto
interessante";
// convertiamo in array
$array = split("\t\n",$stringa);
echo $array[0]; // stampa "Visita"
echo $array[1]; // stampa "il"
echo $array[2]; // stampa "sito"
echo $array[3]; // stampa "html.it"
.....
?>
```

Conclusioni sulle stringhe

Con questo articolo non si è certamente esaurito l'argomento stringhe, basta infatti consultare la documentazione ufficiale alla url <http://www.php.net/manual/en/ref.strings.php>, per rendersene conto. Le funzioni sono tantissime ed è sempre consigliabile uno sguardo a queste pagine quando si debba risolvere un particolare problema: meglio risparmiare codice se esiste già una funzione che fa' al caso nostro. Si sono volutamente trascurate alcune funzioni che coinvolgono l'utilizzo di espressioni regolari, molto sofisticate e potenti, ma che richiederebbero un articolo a parte. Sono state invece privilegiate alcune tra le funzioni più significative nello sviluppo di applicazioni web, con l'obiettivo di fornire una panoramica sulle notevoli possibilità offerte dal linguaggio nella manipolazione di stringhe.

Array (vettori)

In informatica un **array** o **vettore** è una struttura dati complessa, statica e omogenea, usata in multilinguaggi di programmazione e chiaramente ispirata alla nozione matematica di vettore, (o di matrice, nel caso di array bidimensionali). Più precisamente, l'array è in genere classificato come un costruttore di tipo: in altre parole, esso consente di definire nuovi tipi di dati a partire da (come aggregati di valori di) tipi preesistenti.

Si può immaginare un array come una sorta di casellario, le cui caselle sono dette **celle** dell'array stesso. Ciascuna delle celle si comporta come una variabile tradizionale che rappresenta un *elemento* dell'array; tutte le celle sono variabili di uno stesso tipo preesistente, detto **tipo base** dell'array. Si parlerà perciò di tipi come "array di interi", "array di stringhe", "array di caratteri" e così via. Quello che si ottiene dichiarandolo è dunque un contenitore statico ed omogeneo di valori, variabili o oggetti. In alcuni linguaggi, la *dimensione* dell'array (ovvero il numero delle celle di cui esso è composto) viene considerato parte della definizione del tipo array; in tal caso, si parlerà più precisamente di tipi come "array di 100 caratteri" o "array di 10 interi".

L'indice dei vettori

Ciascuna delle celle dell'array è identificata da un valore di **indice**. L'indice è generalmente numerico e i valori che gli indici possono assumere sono numeri interi *contigui* che partono da 0 o da 1. Si potrà quindi parlare della cella di indice 0, di indice 1, e, in generale, di indice N, dove N è un intero compreso fra 0 (o 1) e il valore massimo per gli indici dell'array. La sua principale caratteristica è che può essere *indicizzato* tramite un indice intero e scorso sequenzialmente in entrambe le direzioni tramite un ciclo iterativo in tutti i suoi elementi o a partire da alcuni di essi oltre a poter accedere singolarmente ad una sua generica posizione.

Ecco un esempio, che si serve della sintassi C (simile, comunque, a quella di molti altri linguaggi) per definire e valorizzare un array:

```
int vettore[10]; // definisce "vettore" come array di 10 elementi interi
vettore[0] = 0; // assegna il valore "0" alla cella di indice 0
vettore[1] = 1;
vettore[2] = 1;
vettore[3] = 2;
vettore[4] = 3;
```

(Il vettore sopra indicato contiene i primi cinque numeri di Fibonacci).

Alcuni linguaggi ammettono indici di tipo non numerico, per esempio stringhe. Si parla in questo caso di hash table, o di array associativo, perché ogni valore stringa utilizzato come indice viene associato a un valore dell'array. Vediamo un esempio in linguaggio PHP:

```
$persona["nome"] = "Mario";
$persona["cognome"] = "Rossi";
$persona["eta"] = 32;
```

Come si vede, l'esempio è molto simile al precedente; l'unica differenza rilevante (se si esclude una piccola differenza puramente sintattica tra i linguaggi) è che l'indice dell'array è di tipo stringa. Inoltre, il lettore esperto potrà osservare che in PHP non esiste il vincolo di un "tipo base" fissato per tutte le celle dell'array: alle prima due è stata assegnata una stringa, alla terza un numero intero.

Il PHP supporta sia gli array scalari che gli array associativi.

In PHP, un array di valori può essere esplicitamente creato definendone gli elementi oppure la sua creazione può avvenire inserendo valori all'interno dell'array, ad esempio:

```
$a = array ("qui", "quo", "qua");
```

crea l'array definendo esplicitamente gli elementi dell'array, al contrario dell'esempio che segue:

```
$a[0] = "qui";  
$a[1] = "quo";  
$a[2] = "qua";
```

Se invece, per aggiungere elementi ad un array (supponiamo che sia quello precedentemente creato) si utilizzano le parentesi quadre vuote, i dati vengono accodati all'array; ad esempio:

```
$a[] = "pippo";  
$a[] = "pluto";
```

In questo caso, l'array si allunga di 2 elementi e risulta:

```
$a[0] = "qui";  
$a[1] = "quo";  
$a[2] = "qua";  
$a[3] = "pippo";  
$a[4] = "pluto";
```

Gli array associativi si basano invece su coppie "name-value"; un esempio potrebbe essere:

```
$a = array(  
"nome" => "Mario",  
"cognome" => "Rossi",  
"email" => "mario@rossi.com",  
);
```

E' interessante la possibilità della funziona array di annidare le entries, come nell'esempio che segue:

```
$a = array(  
  "primo" => array(  
    "nome" => "Mario",  
    "cognome" => "Rossi",  
    "email" => "mario@rossi.com",  
  ),  
  "secondo" => array(  
    "nome" => "Marco",  
    "cognome" => "Verdi",  
    "email" => "mario@verdi.com"));
```

Eeguire su questo array un comando del tipo:

```
<? echo $a["secondo"]["email"]; ?>  
visualizzerà "mario@verdi.com"
```

Operatori aritmetici

I principali operatori aritmetici sono i seguenti:

$\$a + \b	Somme	Sum of \$a and \$b.
$\$a - \b	Sottrazione	Difference of \$a and \$b.
$\$a * \b	Moltiplicazione	Product of \$a and \$b.
$\$a / \b	Divisione	Quotient of \$a and \$b.
$\$a \% \b	Resto	Remainder of \$a divided by \$b.

Operatori di confronto

I principali operatori di confronto sono i seguenti:

$\$a == \b	Uguaglianza (TRUE se \$a è uguale a \$b)
$\$a === \b	Identità (TRUE se \$a è identico a \$b, e sono dello stesso tipo (solo PHP 4!))
$\$a != \b	Disuguaglianza (TRUE se \$a non è uguale a \$b)
$\$a <> \b	Come sopra
$\$a !== \b	Non indentità (TRUE se \$a non è identico a \$b, o non sono dello stesso tipo (solo PHP 4))
$\$a < \b	Minore di (TRUE se \$a strettamente minore di \$b)
$\$a > \b	Maggiore di (TRUE se \$a è strettamente maggiore \$b)
$\$a <= \b	Minore o uguale (TRUE se \$a è minore o uguale a \$b)
$\$a >= \b	Maggiore o uguale (TRUE se \$a è maggiore o uguale a \$b)

Operatori logici

I principali operatori logici sono i seguenti:

$a \text{ and } b$ And TRUE se a e b sono TRUE.

$a \text{ or } b$ Or TRUE se a o b sono TRUE.

$a \text{ xor } b$ Xor TRUE se a o b sono TRUE (ma non entrambi).

$!a$ Not TRUE se a non è TRUE.

$a \&\& b$ And TRUE se a e b sono TRUE.

$a || b$ Or TRUE se a o b sono TRUE.

Nota: I due operatori per congiunzione e disgiunzione hanno diversa precedenza.

Strutture di controllo

In tutti i paradigmi di programmazione imperativa, le **strutture di controllo** sono costruiti sintattici di un linguaggio di programmazione la cui semantica afferisce al controllo del flusso di esecuzione di un programma, ovvero servono a specificare *se, quando, in quale ordine e quante volte* devono essere eseguite le istruzioni che compongono il codice sorgente in base alle specifiche di progetto del software da realizzare.

Strutture di controllo fondamentali

Sequenza

La **sequenza** è la struttura di controllo fondamentale di qualsiasi linguaggio imperativo, inclusi i linguaggi macchina. Stabilisce l'ordine in cui le istruzioni presenti nel testo del programma devono essere eseguite a tempo di esecuzione. Di norma, non ha una espressione sintattica esplicita: invece è data dalla semplice successione delle istruzioni; in tal caso, la macchina astratta del linguaggio in uso le esegue in sequenza, una dopo l'altra.

Goto (non è consigliabile l'utilizzo)

Insieme alla sequenza, il **goto** (*vai a*) è la struttura di controllo più semplice; anch'essa appare, in qualche forma, in tutti i linguaggi macchina. Il significato generale del *goto* è quello di "salto" ovvero far "passare" il controllo a una istruzione specificata, che può trovarsi in un punto *qualsiasi* del programma. Il *goto* ammette sempre anche (o solo) una forma *condizionata*, il cui significato può essere parafrasato come segue: "se è vera una condizione *C*, *vai alla* istruzione *I*". Nei linguaggi macchina, la condizione *C* deve solitamente assumere una delle due seguenti forme: "il contenuto della cella di memoria *M* è 0" oppure "il contenuto della cella di memoria *M* è diverso da 0"; l'istruzione *I* viene inoltre identificata dall'indirizzo di memoria in cui *I* è memorizzata.

Nei Linguaggi ad alto livello come il C, la condizione *C* può essere espressa come una qualsiasi espressione booleana (come *il risultato della moltiplicazione di A per B è diverso da X*), e l'istruzione *I* può essere identificata da un nome o da un codice numerico esplicitamente associato dal programmatore all'istruzione stessa (indipendentemente dalla sua collocazione in memoria).

A partire dagli anni settanta, la struttura di controllo *goto* è stata soggetta a forti critiche in quanto essa consente (o favorisce) lo sviluppo di programmi potenzialmente molto poco leggibili e modificabili (il cosiddetto spaghetti code). Sebbene essa resti una struttura di controllo fondamentale dei linguaggi macchina moderni, nei linguaggi di programmazione ad alto livello, a seconda dei casi, il *goto* non viene fornito oppure viene fornito ma se ne sconsiglia l'uso.

Strutture di controllo della programmazione strutturata

Nel 1966, con un celebre teorema, Corrado Böhm e Giuseppe Jacopini introdussero i fondamenti teorici del paradigma della programmazione strutturata dimostrando che qualsiasi programma scritto usando la struttura di controllo goto poteva essere riscritto usando solo strutture di controllo di tipo sequenza, iterazione e alternativa. Unitamente alle critiche di cui si è detto sopra, questo risultato contribuì a decretare la fine della programmazione basata sul goto. Tutti i linguaggi moderni offrono un insieme di strutture di controllo dei tre tipi introdotti da Böhm e Jacopini, sebbene molti mantengano anche il goto (pur sconsigliandone l'uso indiscriminato).

Alternativa

Le strutture di controllo "alternative" consentono di specificare che una data istruzione o un dato blocco di istruzioni devono essere eseguiti "(solo) se" vale una certa condizione. Sono perciò anche dette *strutture condizionali*.

Alternativa if-then e if-then-else

L'alternativa **if-then** (*se-allora*) è la più semplice forma di alternativa. Il suo significato può essere parafrasato con la frase "se vale la condizione *C*, esegui l'istruzione (blocco) *I*". La maggior parte dei linguaggi di programmazione ammette anche (come variante) la forma più articolata **if-then-else** (*se-allora-altrimenti*), che si può parafrasare come: "se vale la condizione *C* esegui l'istruzione (blocco) *I1*; altrimenti esegui l'istruzione (blocco) *I2*".

L'alternativa case

L'**alternativa case** può essere assimilata a una catena di if-then-else con certe restrizioni. In questo caso, la scelta di uno fra *N* istruzioni o blocchi alternativi viene fatta sulla base del valore di una determinata variabile o espressione, normalmente di tipo intero. Essa può essere parafrasata come segue: "valuta il valore *N*; nel caso in cui il suo valore sia *V1*, esegui *I1*; nel caso in cui sia *V2*, esegui *I2* (ecc.)". Il seguente frammento di codice pseudo-C illustra il concetto:

```
case sceltaMenu of
1: apriFile();
2: chiudiFile();
3: salvaFile();
4: esci();
end;
```

Il fatto che il valore *N* debba (spesso) essere di tipo intero è legato a considerazioni di efficienza nell'implementazione del meccanismo. Il *case* si presta infatti a essere trasformato, a livello di linguaggio macchina, in un *goto* con indirizzamento indiretto, che potrebbe essere basato su una tabella in memoria di cui *N* seleziona una particolare entry, in cui è specificato l'indirizzo delle istruzioni da eseguire per quel valore di *N*.

Iterazione

Le strutture di controllo "iterative" consentono di specificare che una data istruzione o un dato blocco di istruzioni devono essere eseguiti ripetutamente. Esse vengono anche dette **cicli**. Ogni struttura di controllo di questo tipo deve consentire di specificare sotto quali condizioni l'iterazione (ripetizione) di tali istruzioni debba terminare, ovvero la *condizione di terminazione del ciclo* oppure, equivalentemente, la *condizione di permanenza nel ciclo*. Di seguito si esaminano le strutture di controllo più note di questa categoria.

Ciclo for

Il **ciclo for** è indicato quando il modo più naturale per esprimere la condizione di permanenza in un ciclo consiste nello specificare *quante volte* debbano essere ripetuti l'istruzione o il blocco controllati dal ciclo. Le forme tradizionali di ciclo for possono essere parafrasate come "ripeti (il codice controllato) per *i* che va da un certo valore iniziale a un certo valore finale, con un certo *passo*". *i* è in generale una variabile di tipo intero, detta **contatore**. Il seguente frammento di codice Basic illustra il concetto:

```
FOR I=1 TO 9 STEP 2
PRINT I
```

Questo frammento ripete l'istruzione di stampa a video della variabile contatore I. Essa parte dal valore iniziale 1 per arrivare al valore finale 10. L'indicazione del "passo" (STEP) specifica come varia il valore di I da una iterazione alla successiva. Il frammento dunque stamperà la sequenza 1, 3, 5, 7, 9.

Ciclo while

Il **ciclo while** (*mentre, o fintantoché*) è indicato quando la condizione di permanenza in un ciclo è una generica condizione booleana, indipendente dal numero di iterazioni eseguite. Le forme tradizionali di ciclo while possono essere parafrasate come "ripeti (il codice controllato) fintantoché resta vera la condizione *C*". Un esempio tipico è la lettura di dati da un file di cui non si conosca a priori la dimensione; esso potrebbe assumere la forma "leggi il prossimo dato finché non incontri la fine del file". Il seguente frammento di codice pseudo-C mostra un esempio di *while*:

```
passwordUtente = leggiPassword();
while(passwordUtente <> passwordCorretta) {
    segnalaErrorePassword();
    passwordUtente = leggiPassword();
}
```

Nel *while* del C (e di molti altri linguaggi) la condizione di permanenza nel ciclo viene controllata prima di eseguire la prima iterazione del ciclo stesso; se essa risulta immediatamente falsa, le istruzioni nel ciclo *non vengono eseguite*. Nell'esempio riportato sopra, se la password letta è corretta, il blocco di codice che segnala l'errore di immissione e chiede di inserire nuovamente la password non viene (ovviamente) eseguito.

Ciclo loop-until

Il **ciclo loop-until** (*ripeti finché*) differisce dal while per due aspetti. Innanzitutto, esso garantisce che venga eseguita sempre almeno una iterazione del ciclo (la condizione che controlla il ciclo viene verificata *dopo* aver concluso la prima iterazione). In secondo luogo, viene espressa la *condizione di terminazione del ciclo* anziché quella di *permanenza nel ciclo* (quest'ultima differenza non ha comunque un impatto concettuale molto

importante, essendo le due condizioni esprimibili semplicemente come negazione l'una dell'altra). Il seguente frammento pseudo-Pascal illustra il concetto:

```
loop
passwordUtente:= leggiPassword();
until (passwordUtente = passwordCorretta)
```

Varianti di while e loop-until

Le due differenze citate sopra fra *while* e *loop-until* sono in effetti indipendenti l'una dall'altra, per cui sono facilmente immaginabili (benché meno diffuse per motivi storici) anche altre due combinazioni: cicli che garantiscono una iterazione ma in cui si specifica la condizione di permanenza nel ciclo, e cicli che ammettono o iterazioni ma in cui si specifica la condizione di terminazione del ciclo. Un esempio del primo caso è la struttura **do-while** (*fai fintantoché*) del C e dei suoi derivati, esemplificata in questo frammento di pseudo-codice:.

```
do
passwordUtente = leggiPassword();
while(passwordUtente<>passwordCorretta);
```

Iterazione basata su collezioni

Alcuni linguaggi (per esempio Smalltalk, Perl, C#, Java) forniscono varianti del ciclo for in cui il "contatore" è una variabile generica (non necessariamente un numero intero) che assume una sequenza di valori del tipo corrispondente, per esempio tutti i valori contenuti in un array o in una collezione. Il seguente frammento di codice C# illustra il concetto:

```
foreach (string s in unaCollezioneDiStringhe) { stampa(s); }
```

Terminazione anticipata di cicli e iterazioni

Molti linguaggi forniscono una istruzione specifica per terminare "prematuramente" un ciclo, ovvero causarne la terminazione in un modo alternativo rispetto a quello principale del ciclo (per esempio basato sul valore del contatore nel *for* o sulla verità/falsità di una condizione nei cicli *while* o *repeat-until*). Il seguente frammento illustra il concetto in un ciclo Java:

```
// cerco un valore N in un array
boolean trovato = false;
for(int i=0; i<100; i++)
if(v[i]==N) {
    trovato = true;
    break;
}
```

In questo frammento Java, il programma deve stabilire se un dato numero N è contenuto in un array. Non appena N viene trovato, diventa inutile proseguire l'attraversamento dell'array stesso: il *break* termina quindi il ciclo *for*.

L'uso indiscriminato di meccanismi come il *break* (fornito da linguaggi come C, C++ e Java) è spesso criticato e sconsigliato perché si viene a perdere una utile proprietà di leggibilità dei cicli della programmazione strutturata: infatti, di fronte a un ciclo *while* con una condizione *C*, il lettore tende ad assumere che, al termine del ciclo, *C* sia falsa. L'uso di *break* o costrutti simili, introducendo "un altro" possibile motivo di terminazione del ciclo (oltretutto potenzialmente "nascosto" all'interno del blocco controllato dal *while*) fa sì che questa assunzione non del tutto sicura. Tuttavia, vi sono anche casi in cui lo stato del programma alla fine del ciclo, con o senza *break*, è lo stesso, per cui il suddetto problema non si pone. Questo è il caso dell'esempio Java riportato sopra (il contatore *i* è locale al *for*, e quindi viene deallocato al termine del ciclo).

Un meccanismo simile (in generale considerato meno pericoloso) è quello che consente di terminare anticipatamente *una determinata iterazione* di un ciclo, passando immediatamente alla successiva. Si veda il seguente programma:

```
for(int i=1; i<=100; i++)
if(numeroPrimo(i)) {
    stampa(i);
    continue;
}
// calcola e stampa i fattori di i
}
```

Questo programma stampa i fattori di tutti i numeri interi da 1 a 100. Se *i* è un numero primo, è sufficiente stampare il numero stesso; altrimenti sono necessarie altre operazioni per scomporlo.

L'istruzione *continue* indica che l'iterazione corrente è conclusa e fa sì che il ciclo passi immediatamente alla successiva.

- if - endif
 - else
 - elseif
 - while
 - for-each
 - switch
-

if (in Php)

If permette di eseguire un blocco di codice se avviene (o non avviene) una determinata condizione; la sua sintassi è:

```
if (condizione) {  
    statement }
```

Ad esempio:

```
$a = 2;  
$b = 2;  
if ($a == $b) {  
    echo "\$a è uguale a \$b e valgono $a.\n";  
}
```

endif if (in Php)

If può essere usato anche senza graffe se si utilizza il comando “endif”:

Ad esempio:

```
$a = 2;  
$b = 2;  
if ($a == $b)  
echo "\$a è uguale a \$b e valgono $a.\n";  
endif;
```

if-else if (in Php)

“else” viene in completamento di “if”: con if, infatti, stabiliamo che succeda qualcosa all'avverarsi di una condizione; con else possiamo stabilire cosa accade nel caso questa non si avveri. Un esempio potrebbe essere:

```
$a = 2;  
$b = 3;  
if ($a == $b) {  
    echo "\$a è uguale a \$b e valgono $a.\n";  
} else {  
    echo "\$a è diversa da \$b.\n$a vale $a mentre $b vale $b.\n";  
}
```

if-elseif-else if (in Php)

elseif permette di specificare casi non definiti da “if”:

```
if ($a == $b) {  
    echo "\$a è uguale a \$b.\n";  
}  
elseif ($a != $b) {  
    echo "\$a è diversa da \$b.\n";  
}
```

```
}
elseif (!$a) {
    echo "\$a non esiste.\n";
}
elseif (!$b) {
    echo "\$b non esiste.\n";
}
else {
    echo "Non so che dire ... \n";
}
```

while if (in Php)

La condizione "while" si comporta esattamente come in C; la sintassi di base è:

```
while (espressione) statement
```

Per esempio:

```
/* $a viene incrementato e visualizzato */
/* finchè il suo valore non supera "5" */
$a = 1;
while ($a <= 5) {
    print $a++;
}
```

for if (in Php)

Il "for" permette di eseguire dei loop. La sintassi è la seguente:

```
for (expr1; expr2; expr3) statement
```

Per esempio:

```
for ($a = 0 ; $a <=10 ; $a++) {
    print $a;
}
```

visualizzerà i numeri da "0" a "10". Nelle tre espressioni fra parentesi abbiamo definito che:

\$a ha valore "0" (valutato una sola volta all'inizio del loop);

\$a è minore o uguale a "10";

\$a è incrementata di un'unità.

Quindi, per ogni valore di \$a a partire da "0" fino a "10" \$a viene visualizzato.

Foreach if (in Php)

Il PHP 4 permette di eseguire loop su array in modo semplificato usando il costrutto "foreach". La sintassi è la seguente:

```
foreach (array_expression as $value) statement
```

```
foreach (array_expression as $key => $value) statement
```

Per esempio:

```
$arr = array("one", "two", "three");
```

```
foreach ($arr as $value) {  
    echo "Value: $value<br>\n";  
}
```

```
/* Esempio: key and value */
```

```
$a = array ( "one" => 1, "two" => 2, "three" => 3, "seventeen" => 17 );
```

```
foreach($a as $k => $v) {  
    print "\$a[$k] => $v.\n"; }
```

```
/* Esempio: arrays multi-dimensionali*/
```

```
$a[0][0] = "a";
```

```
$a[0][1] = "b";
```

```
$a[1][0] = "y";
```

```
$a[1][1] = "z";
```

```
foreach($a as $v1) {  
    foreach ($v1 as $v2) {  
        print "$v2\n";  
    }  
}
```

switch-case-default if (in Php)

"Switch" permette di sostituire una serie di "if" sulla stessa espressione e, ovviamente, di agire dipendentemente dal valore di questa:

```
switch($a) {  
  case 1:  
    print("a è uguale a uno");  
    break;  
  case 2:  
    print("a è uguale a due");  
    break;  
  default:  
    print("a non vale né uno né due");  
}
```


Form HTML: sintassi

Un form HTML è una finestra contenente vari elementi di controllo che consentono al visitatore di inserire informazioni.

Una volta inseriti, i dati vengono inviati ad uno script che li elabora.

Sintassi:

```
<form action="<action="[URL dello script][script]" method="method" method="[GET o POST]"="[POST]">>  
<input type="<type="[elemento di controllo][controllo]" name="name" name="[nome]"="[nome]" ...>" ...>  
<input type="<type="[elemento di controllo][controllo]" name="name" name="[nome]"="[nome]" ...>" ...>  
.....  
</form></form>
```

FORM HTML

Immaginiamo di avere la seguente form HTML:

```
<FORM action="http://localhost/password.php" method="post">  
<INPUT type="TEXT" name="utente">  
<INPUT type="SUBMIT" name="Invia" value="Spedisci">  
</FORM>
```

L'attributo action serve per specificare l'URL dello script.

L'attributo method serve per specificare la modalità di invio delle informazioni. Può essere GET o POST. Con GET le informazioni vengono inserite nell'indirizzo URL, dunque sono visibili nella barra degli indirizzi del browser, ma sono vincolate dalla lunghezza massima di un URL, che è di 256 caratteri. Con POST i dati vengono scritti sullo "standard input" dell'applicazione destinataria, dunque non sono visibili ma soprattutto non ci sono limiti sulla quantità di dati inviata.

Si può accedere alle informazioni inviate anche utilizzando gli array superglobali \$_GET e \$_POST (a seconda del metodo usato).

Accedere alle variabili di una FORM HTML

Ci sono vari modi di accedere a variabili da una form HTML. Per esempio, se si utilizza il metodo POST, ci sono i seguenti modi:

```
<?php
// Available since PHP 4.1.0
print $_POST['utente'];
print $_REQUEST['utente'];
import_request_variables('p', 'p_');
print $p_utente;

// Available since PHP 3.
print $HTTP_POST_VARS['utente'];

// Available if the PHP directive register_globals = on. As of
// PHP 4.2.0 the default value of register_globals = off.
// Using/relying on this method is not preferred.
print $utente; ?>
```

Esempio

```
<?php
if (isset($_POST[utente]) {
    echo "Benvenuto <b>$_POST[utente]</b>, divertiti";
}
else {
    echo "Torna indietro e inserisci il nome utente!";
}

?>
```

Inviare una mail in PHP e gestire le form

Prerequisiti

Per seguire questa lezione occorrono dei prerequisiti:

1) conoscere i tag html relativi ai moduli (da qui in poi li chiameremo, indistintamente, "form"). Ovvero, sapere come creare un campo testuale, un input di password, una comboBox (la "select"), i pulsanti di scelta multipla o singola (check/radio button).

2) conoscere la sintassi base di PHP (se avete seguito almeno le prime lezioni di questo corso non dovrete aver problemi)

3) avere uno spazio web (online) dove testare lo script. Anche se in locale potete eseguire i test, quando si tratta di inviare email dovete comunque installare e configurare anche un server di posta in uscita (SMTP). L'operazione non è semplice, vi consiglio decisamente di aprire uno spazio web (anche uno gratuito come altervista va bene) e testare online gli script.

Potreste anche modificare il file php.ini impostando l'SMTP con cui vi connettete ad internet, ma non tutti i provider accettano invii anonimi di email e comunque dovrete collegarvi all'accesso remoto. Tanto vale utilizzare uno spazio web gratuito.

Interagiamo con i moduli (form)

Per inviare una email in PHP, in questa lezione, utilizzeremo due pagine:

.form.html (un normale file html contenente il codice per disegnare il modulo di inserimento dati);

.elabora_form.php (un file php contenente il codice che invia l'email e restituisce il messaggio di "email inviata con successo" all'utente)

Per il file "form.html" non credo ci siano problemi, attenzione però al tag <form>. Finora (se non avete altre esperienze in linguaggi dinamici) avete utilizzato questo tag per consuetudine, al massimo per richiamare un evento Javascript di controllo dati inseriti:

```
<form name="modulo" onSubmit="return controlloJavaScript()">
```

Con PHP, possiamo aggiungere (e togliere) alcune proprietà al tag. Intanto il method:

```
<form name="modulo" onSubmit="return controlloJavaScript()" method="post">
```

Il "method" è il modo con cui vengono inviate le informazioni. Un "method" impostato a

"GET" aggiunge alla url tutti i dati scritti nei campi del modulo.

Ipotizziamo di avere una form con due campi del genere:

Nome:

Password:

Impostando il method a get, ed ipotizzando come nome "claudio" e come password "curci" (qualcosa a prova di lamer in pratica :), avremo nella pagina successiva una url di questo tipo:

`elabora_form.php?nome=claudio&password=curci`

dove "nome" e "password" sono i nomi dei due campi (attributo "name=", visualizzate l'HTML della pagina per chiarire ogni dubbio).

Questo modo (metodo) di scambio informazioni tra due pagine HTML è poco sicuro (mai sentito parlare di attacchi "brute force", ovvero di software che inviano ad una pagina web - php o asp che sia- tutte le combinazioni possibili di lettere?) e limitato (la url non può contenere più di 255 caratteri).

Il passaggio di variabili dalla url può essere comodo per i link (nelle prossime lezioni vedremo alcune funzioni in merito), come "`a href=nomePagina.php?argomento=viaggi`". Ad una stessa url (nomePagina.php) viene associato un contenuto diverso a seconda della variabile passata come "argomento", sia "viaggi" o "sport" o "politica".

Per le form è bene utilizzare un altro metodo, che non accodi le variabili alla url della pagina: il POST.

Nel caso precedente, un action con method impostato a POST non produce altro che la url della pagina:

`elabora_form.php`

I dati vengono passati in modo trasparente all'utente.

Detto questo, possiamo disegnare la nostra form. E' importante associare ad ogni campo un nome. Tenete presente che ciascun nome che daremo ai campi diventerà, nella pagina successiva, una variabile PHP. Quindi utilizzate nomi di facile memoria e tenete presente le regole delle variabili (non utilizzate spazi, non iniziate il nome con un numero, lasciate i caratteri "strani" come i punti e le virgole a casa..)

Esempio di file "form.html" (il nome è ovviamente arbitrario):

```

<form name="modulo" action="elabora_form.php" method="post">
<table>
  <tr>
    <td>Nome utente:</td><td><input type="text" name="nome" si
  </tr>
  <tr>
    <td>Indirizzo email:</td><td><input type="text" name="indirizz
  </tr>
  <tr>
    <td>Motivo del contatto:</td>
    <td>
      <select name="motivo">
        <option value="commerciale">Informazioni commerciali</option>
        <option value="preventivo">Richiesta preventivo</option>
        <option value="appuntamento">Prenotazione appuntamento</option>
      </select>
    </td>
  </tr>
  <tr>
    <td>Note:</td><td><textarea name="testo" cols="20" row s="7
  </tr>
  <tr>
    <td colspan="2" align="center"><input type="submit" value="In
  </td>
</tr>
</table>
</form>

```

Ovviamente il codice della tabella è opzionale, serve solo per impaginare meglio il modulo. Il risultato finale è questo:

Nome utente:

Indirizzo email:

Motivo del contatto:

☐ Informazioni commerciali
 ☐ Richiesta preventivo
 ☐ Prenotazione appuntamento

Note:

Scrivi NoSpam nella casella qui sotto:

Un normale form HTML (lasciate stare il colore di sfondo o il bordo, quelli dipendono dal CSS di questo corso!) vero?

Passiamo alla gestione del modulo. Create un normale file con il vostro editor, dentro inserite i tag fondamentali (head, body..) e salvatelo come elabora_form.php, nella stessa directory di "form.html" (per ora potete lavorare in locale sulla root di easyPHP/XAMPP, quando invieremo la mail ci trasferiremo online).

Se (via browser) accedete al file della form, compilate i campi e cliccate sul tasto di invio, aprite la pagina php di elaborazione e non trovate nulla. E' necessario dare all'interprete php i comandi per interagire con la form.

Abbiamo visto prima che, quando inviamo i dati di un modulo ad una pagina PHP, abbiamo automaticamente a disposizione delle variabili il cui nome è lo stesso dei campi. Inserite nella pagina elabora_form.php questo codice, che stamperà a video il valore dei campi della form.

```

Riepilogo campi:
<?php
    echo "nome: " . $_POST['nome'] . " <br>";
    echo "indirizzo: " . $_POST['indirizzo'] . " <br>";
    echo "motivo del contatto: " . $_POST['motivo'] . " <br>";
    echo "testo inserito: " . $_POST['testo'];

?>

```

Adesso (lanciate prima EasyPhp/XAMPP se non l'avete già fatto), puntate il vostro browser sulla pagina form.html e compilate i campi del modulo a vostro piacere:

Indirizzo <http://127.0.0.1/corso/form/form.html>

Nome utente:

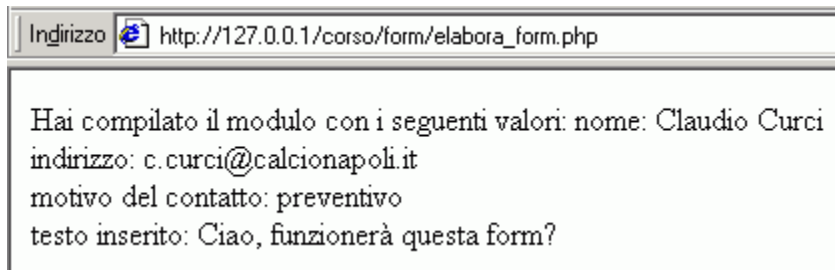
Indirizzo email:

Motivo del contatto:

Note:

Cliccate su "invia email" e, se avete compilato e salvato correttamente il file

elabora_form.php, dovrete avere un output del genere:



Abbiamo visto che una variabile può essere passata con method POST ma anche con GET. Nell'esempio sopra, un utente potrebbe tranquillamente richiamare la pagina "elabora_form.php" passando i valori nella url, senza passare per la form. Digitate nella url del browser le variabili (i nomi dei campi della form) assegnando dei valori. Le variabili vanno separate con la "&", secondo questa sintassi:

nomeFile.php?var1=valore1&var2=valore2

digitate qualcosa del genere (senza passare per form.html):

http://127.0.0.1/corso/form/elabora_form.php?nome=alessio&indirizzo=info@info.com&motivo=nessuno&testo=visto che ti volevo fregare?

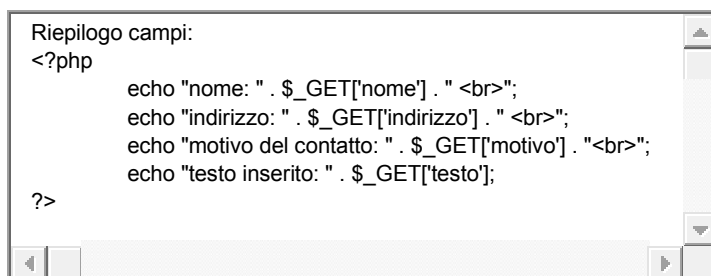
(ovviamente la parte "corso/form/" è relativa alle cartelle che utilizzo io nel computer, se avete salvato tutto direttamente dentro la root basta scrivere http://127.0.0.1/elabora_form.php?ecc.)

A video non comparirà nulla, poiché abbiamo usato un metodo diverso di invio dei dati.

E' opportuno quindi distinguere le variabili provenienti da richieste GET da quelle POST. PHP mette a disposizione degli appositi array: \$_POST[] e \$_GET[].

Funzionano come dei normali array associativi, e sono sicuri poiché le variabili passate nella url (GET) sono in un contenitore, quelle con POST in un altro.

Se volete provare a passare i valori attraverso la URL e quindi in modalità GET, dovrete cambiare l'esempio sopra in questo modo:



Se avete seguito la lezione sugli array associativi, credo non abbiate bisogno di commenti sui

codici sopra: richiamiamo infatti una variabile dell'array `$_POST[]` o `$_GET[]` con il relativo indice, che corrisponde al nome del campo.

Dobbiamo ricordarci che i nomi degli indici degli array associativi vanno sempre racchiusi tra apici (singoli o doppi).

la funzione `include()`

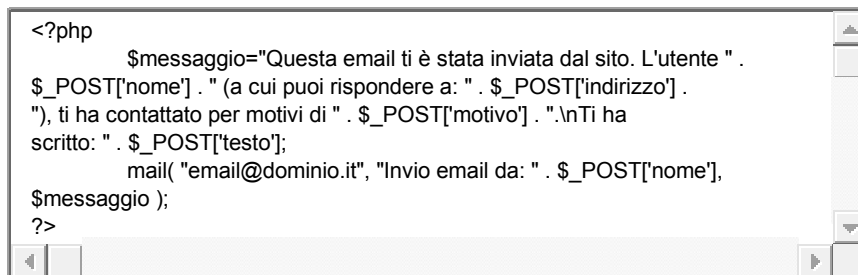
Andiamo avanti e finalmente creiamo qualcosa di utile da inserire subito nel nostro sito: la possibilità di farci inviare email personalizzate!

PHP mette a disposizione una funzione, guardacaso chiamata `mail()` che riceve 4 parametri (l'ultimo è opzionale):

- destinatario;
- oggetto;
- messaggio;
- header aggiuntivi

Quello che vogliamo è quindi prendere i dati della form di prima (sappiamo già come farlo) e inserirli in quei parametri.

Teniamo la pagina `form.html` come è adesso e cambiamo la pagina `elabora_form.php` in questo modo:



```
<?php
    $messaggio="Questa email ti è stata inviata dal sito. L'utente " .
    $_POST['nome'] . " (a cui puoi rispondere a: " . $_POST['indirizzo'] .
    "), ti ha contattato per motivi di " . $_POST['motivo'] . ".\nTi ha
    scritto: " . $_POST['testo'];
    mail( "email@dominio.it", "Invio email da: " . $_POST['nome'],
    $messaggio );
?>
```

(solito discorso, andate a capo solo quando vedete il punto e virgola: in questo caso abbiamo due sole righe ma per motivi di impaginazione la textarea va a capo e sembrano 3).

A questo punto però, per testare lo script, dovete pubblicare su uno spazio web (anche uno gratuito va bene) le due pagine: `form.html` e `elabora_form.php`.

Ovviamente sostituite l'indirizzo di posta indicato con il vostro nel primo parametro, altrimenti inviereste tutte le mail ad un indirizzo inesistente e non sapreste mai se lo script è andato a buon fine :)

Se tutto è andato ok, potete divertirvi configurando meglio la variabile `$messaggio`. La mail però è un pò bruttina vero? Il fatto è che viene inviata in normale formato testo (infatti per andare a capo abbiamo utilizzato lo `"\n"`, non il tag `
`).

Anche se i "guru" di internet consigliano di inviare email esclusivamente in formato testo (per ovvi motivi di sicurezza, il txt è innocuo) e la "netiquette" fa altrettanto (una mail in html è comunque più pesante), è bene vedere come inviare attraverso PHP mail HTML. Del resto, se ci occupiamo di un sito di e-commerce, l'invio di email html non viola alcuna netiquette (tutto va a finire nella casella del

cliente, se sta bene a lui sta bene a tutti no?) ed è anzi necessaria (immaginate di dover intabellare un'ordine di 20 prodotti con relativi prezzi, sconti, aliquote iva in un file txt..)

Abbiamo parlato prima del quarto parametro, ovvero gli "header" addizionali. Si tratta di informazioni non obbligatorie ma che aumentano la funzionalità del comando.

Possiamo, ad esempio, decidere il mittente (omettendo il parametro apparirà infatti, nella casella email del destinatario, un messaggio proveniente da qualcosa come "httpd" o comunque un header deciso dal provider); così come possiamo decidere il formato del messaggio.

La sintassi del parametro "header opzionali" è particolare. Occorre separare le varie intestazioni (mittente, formato ecc.) con il carattere di "a capo" \r\n.

Rivisitiamo lo script precedente con l'istruzione di formattazione html. Ovviamente, dobbiamo inserire nel messaggio anche i vari tag (che ben conosciamo), proprio come se ci trovassimo su una pagina web.

```
<?php
    $intestazioni = "MIME-Version: 1.0\r\n";
    $intestazioni .= "Content-type: text/html; charset=iso-8859-1\r\n";
    //intestazioni per il mittente
    $intestazioni .= "From: miosito<info@miosito.com>\r\n";

    $messaggio="<html><head><title></title></head><body>";
    //non occorre specificare attributi per il tag title, tuttavia li
    inseriamo per correttezza. Tenete presente che potete inserire style.
    $messaggio.="<font face=\"verdana\" size=\"2\">Questa email ti è
    stata inviata dal sito.<br> L'utente " . $_POST['nome'] . " (a cui
    puoi rispondere a: <a href=\"mailto:" . $_POST['indirizzo'] .
    "\"> " . $_POST['indirizzo'] . "</a>), ti ha contattato per
    motivi di " . $_POST['motivo'] . ".<br>Ti ha scritto: " .
    $_POST['testo'] . "</font>";
    $messaggio.="</body></html>";
    mail( "email@dominio.it", "Invio email da: " . $_POST['nome'],
```

Interagiamo con i moduli (form)

Lo script sopra proposto è funzionale ma è bene, fin d'ora, migliorare ed affinare le nostre capacità di programmazione.

Come vedete, modificare le istruzioni php è operazione delicata. Immaginate se un grafico aprisse il vostro script php con DreamWeaver (proprio la scorsa settimana ho dovuto mettere mano ad un codice che il programma Adobe aveva "gentilmente" imbastardito togliendo tutti gli "a capo" e le tabulazioni..) e cercasse di modificare parametri come il colore di sfondo della mail o il font del testo..

Onde evitare spargimenti di sangue, è bene utilizzare un paio di accorgimenti semplici ma efficaci.

Intanto separiamo i parametri dallo script. Creiamo una variabile "\$colore_sfondo", assegnando l'esadecimale desiderato. Stesso discorso per il font e per gli altri stili. In questo modo, se qualcuno vuole modificare lo script o se vogliamo riutilizzarlo in futuro, basta cambiare un'unica riga di codice esterna alle istruzioni e quindi poco pericolosa:

```

<?php
    $colore_sfondo="#82C0FF";
    $colore_testo="#FFFF80";
    $font="verdana";
    $font_size=2;
    $intestazioni = "MIME-Version: 1.0\r\n";
    $intestazioni .= "Content-type: text/html; charset=iso-8859-1\r\n";
    //intestazioni per il mittente
    $intestazioni .= "From: miosito<info@miosito.com>\r\n";

    $messaggio="<html><head><title></title></head><body
    bgcolor=\"\" . $colore_sfondo . "\">"; //non occorre specificare
    attributi per il tag title, tuttavia li inseriamo per correttezza.
    Tenete presente che potete inserire style.
    $messaggio.="<font face=\"\" . $font . "\" size=\"\" . $font_size .
    "\" colore=\"\" . $colore_testo . "\">Questa email ti è stata inviata
    dal sito.<br> L'utente " . $_POST['nome'] . " (a cui puoi
    rispondere a: <a href=\"mailto:\" . $_POST['indirizzo'] . "\"> .
    $_POST['indirizzo'] . "</a>), ti ha contattato per motivi di " .
    $_POST['motivo'] . "<br>Ti ha scritto: " . $_POST['testo'] .
    "</font>";
    $messaggio.="</body></html>";
    mail( "email@dominio.it", "Invio email da: " . $_POST['nome'],

```

Parlavamo però prima del famigerato DreamWeaver, autentico incubo dei programmatori e delle aziende (costrette da alcuni grafici a pagare fior di euro per un programma che crea lo stesso codice ottenibile con notepad). Se avete colleghi che utilizzano prodotti Adobe, è bene che proteggiate il vostro lavoro, frutto magari di pomeriggi e notti passate dietro questa guida (nel forum qualcuno parlava di script realizzati all'una di notte..)

PHP offre una funzione, `include()` che permette di inserire un file nel punto in cui viene chiamato. Una funzione del genere, a ben vedere, può sostituire il tag HTML `<frame>`: basta creare l'intestazione e la barra dei menù in un file a parte e richiamarla ogni volta con `include("nome_intestazione.html")`.

Come compito per questa settimana, esplorate questa funzione e divertitevi nel creare un sito di almeno 5 pagine con logo, menù di navigazione di sinistra e intestazioni di fine pagina distinti in tre file diversi e richiamati da ciascuna pagina.

Fate poi finta di avere la testa di un tipico cliente web, che fino alle 22 ti massacrava con la sua idea di avere il sito con sfondo "red" e l'indomani alle 9 vuole cambiarlo in "green". Cambiate le vostre pagine (basta modificare soltanto quei file per avere tutti gli altri 5 di conseguenza, se avete strutturato bene il discorso) e scoprirete la potenza degli `"include()"`.

Infine, chiudiamo la lezione usando proprio gli `include()` per parametrizzare il nostro esempio.

Se avete codificato il passaggio precedente (creato le famose variabili `$colore_sfondo` e `$font`) potete eseguire un semplice Taglia / Incolla in un nuovo file, chiamato magari "parametri.php" e sostituire quelle istruzioni con un:

```
include("parametri.php");
```

Lo scopo di questo "scorporo" è sia quello di facilitare il lavoro ad altri colleghi, sia di favorire

la leggibilità del codice (immaginate un giorno di creare un intero portale o sistema di e-commerce: avrete decine di variabili comuni a tutte le pagine).

Alcuni usano chiamare i file inclusi con estensione ".inc" ("include"): parametri.inc. E' una consuetudine nota nell'ambiente della programmazione lato client, con linguaggio tradizionali.

NON utilizzate questo approccio sul web perchè è possibile accedere ai file .inc anche via browser. Se considerate che alcuni memorizzano dati come le password di connessione ad un database mysql in uno di questi file, capite quanto sia facile "bucare" un sito - magari passando variabili GET nella url...

FormMail Multiplo in php: come comporre e gestire i vari campi di un modulo

Questa guida propone un modulo di contatto interamente personalizzabile con i campi che si possono aggiungere togliere e cambiare a piacimento.

Il modulo è già pronto da scaricare e ha la caratteristica:

- Di verificare la validità dell'indirizzo e-mail inserito nel campo;
- Di permettere il reindirizzamento dell'utente ad una determinata pagina al termine dell'invio dei dati (nei passaggi successivi vedremo in quale modo).

Il modulo in php è costituito da due file: uno nominato **formmailPhpMultiplo.htm**, ed un secondo file nominato **formInvio.php**.

Vediamo come procedere con la personalizzazione grafica e la configurazione dei moduli (con la possibilità di rendere alcuni campi obbligatori e/o opzionali).

A seguire i semplici passaggi:

1 – Effettuare il download del file.zip



2 - FILE HTML:

Aprire il file con un qualunque editor di testo e/o web editor per **rinominare, aggiungere o rimuovere** alcuni dei campi già presenti nel pacchetto scaricabile.

Consigliamo ai web designer poco esperti, di limitarsi, se necessario a rinominare semplicemente i campi già presenti.

Vedasi immagine che segue (esempio campo “nickname”):

```
top" align="center" bgcolor="#669900">
e="2" color="#FFFFFF">nickname</font></td>
="#CF9F70">
t face="Arial" size="12" color="#993300">&nbsp;
atita.GIF" width="32" height="32"><input type="text" name="nickname"
yle: solid; border-color: #993300"></font>
```



3- FILE PHP

Aprire il file con un qualunque editor di testo e/o web editor e impostare:

Nella RIGA 15: modificare l'indirizzo mail di destinazione (casella di posta dove verranno recapitati i messaggi inviati dagli utenti);

Nella RIGA 20: inserire l'indirizzo del dominio sul quale risiederà il **Modulo php**. Vedasi immagine che segue:

```
//email di destinazione, mettere qui la propria email

$contenitore = "nomecasella@vostramail.xxx";

// Il parametro $provenienza indica le possibili provenienze dei dati: indicare
// il proprio dominio nella forma mostrata dall'esempio. l'IP è facoltativo.

$provenienza = array ('nomedominio.xxx', 'www.nomedominio.xxx', 'xx.xxx.xxx.xxx');
```



Nella RIGA 47: decommentare la riga di codice, eliminando il simbolo "//", e digitare l'indirizzo della pagina alla quale si desidera reindirizzare l'utente una volta terminato l'invio del messaggio tramite il formMail.

Vedasi immagini:

```
44 $esclusioni = array ('*@quellochetipare.com',
45
46 // $redirect = "http://www.masterbass.com";
47
48 $versione_form = "stabile";
```



```
44 $esclusioni = array ('*@quellochetipare.com',  
45  
46 $redirect = "http://www.nomedominio.xxx";  
47  
48 $versione_form = "stabile";
```

IMPOSTAZIONI FACOLTATIVE

Nella RIGA 36: è possibile modificare il tempo (misurato in secondi) entro il quale sarà visualizzata la pagina scelta:

```
$delay = "5";  
  
// Il parametro $esclusioni vi permette di NON consentire  
// messaggi da un indirizzo mail specificato  
// sia appartenente ad un dominio, ovvero  
// 'tutte le mail che appartengono ad un dominio'  
// od anche a singoli account
```

RIGA 17: Per quanto riguarda i campi obbligatori, nel presente esempio sono stati configurati solamente i campi **nome**, **cognome** ed **e-mail**.

Sarà comunque possibile aggiungerne di nuovi inserendo, nell'apposita riga del file in html, i nomi dei campi da rendere obbligatori, come illustrato nell'esempio sottostante.

```
=====
```

<input type="hidden" name="require" value="e-mail,nome">

```
=====
```

4 – PUBBLICAZIONE DEL MODULO PHP:

Una volta ultimata la configurazione e la modifica dei files, salvarli e pubblicarli nello spazio remoto del proprio dominio. Il risultato sarà lo stesso mostrato nell'immagine seguente.

Nell'esempio dimostrativo che segue, sono stati esclusi dall'invio di messaggi tutti i clienti che inseriscono un qualunque indirizzo e-mail appartenente al dominio *mio dominio.it, oltre che tutti coloro che inseriscono un indirizzo del tipo: test@test.com

Nome e Cognome (*)

Nickname

Email
a cui si vuole risposta (*)

Telefono

Messaggio

Opzionale

Opzionale2 funziona?

Opzione3 ☒ desidero essere ricontattato

Opzione4

* campi obbligatori

A seguire i dati che giungono al web designer in seguito alla compilazione del modulo:

Indirizzo email: è indicato nel mittente

Nome e Cognome: link bruttocane

Nickname: link

Telefono: +39 XXX.5X55XXX

Messaggio: prova modulo di contatto

Opzionale: tutto bene?

Opzionale2: funziona?

Contatto: sì

ListaDiscesa: scelta2

IMPORTANTE: in caso di **errore 'proviene da un dominio non autorizzato'**, assicurarsi di aver impostato correttamente il parametro provenienza.

In caso sia correttamente settato assicurarsi di aver configurato le impostazioni di un eventuale firewall sul vostro computer, ad esempio zoneAlarm® per il quale ecco le impostazioni

I file di testo

In informatica il termine **file** (termine inglese per "archivio", ma comunemente detto anche "documento") indica un contenitore di informazioni/dati in formato digitale, tipicamente presenti su un supporto digitale di memorizzazione. Le informazioni scritte/codificate al suo interno sono leggibili solo attraverso software.

Tecnicamente, i dati codificati in un file sono organizzati come una sequenza (di byte), immagazzinati come un solo elemento su una memoria di massa attraverso il File system (sistema di archivi) esistente su quella data memoria di massa.

Ciascun file è identificato da un nome univoco, un'estensione ed un percorso (*path*) che ne individua posizione, contenitore, cartella o directory in uno spazio di nomi gerarchico all'interno del file system stesso.

Se dal punto di vista dell'utente un file è solitamente un singolo elemento, può succedere invece che fisicamente sia scritto o risieda su più zone diverse del supporto di memorizzazione che lo ospita: questo fenomeno è molto comune se il supporto di memorizzazione è un disco di memoria, mentre è molto raro su nastri magnetici. Uno dei compiti del sistema operativo è rendere trasparente alle applicazioni la reale suddivisione fisica del file e occuparsi di gestire il recupero delle informazioni in esso contenute (*lettura*) dai vari blocchi fisici del supporto di memorizzazione e la *scrittura*.

Il contenuto dei file è normalmente conforme ad un particolare formato, e per ciascun formato esistono una o più applicazioni che sono in grado di interpretarne e/o di modificarne il contenuto ("aprire" il file).

Alcuni sistemi operativi, come Microsoft Windows e l'MS-DOS, riconoscono il formato di un file in base all'estensione del loro nome; altri, come il Mac OS, da una serie di metadati salvati insieme al file; altri ancora, come Unix, identificano i tipi di file in base ai primi byte del loro contenuto, detti magic number.

Un tipo di file molto comune sono i file di testo ASCII. Un simile file è una sequenza di caratteri ASCII pura e semplice, tale da poter essere letta ed interpretata nello stesso modo da tutti i sistemi operativi. Il termine "file binario", invece, si riferisce solitamente a tutti i file che non sono di testo ASCII.

Il concetto di file è molto semplice ed elegante: in ultima analisi, un file è una sorgente (o un deposito) di informazioni, che si può leggere e scrivere; questa sorgente/deposito ha anche delle "proprietà" (nome, estensione, flag) che possono essere modificate. Questa definizione si adatta molto bene a molte periferiche ed interfacce hardware; i sistemi operativi Unix e derivati (Unix-like) hanno generalizzato il concetto di file tanto da farne una vera filosofia: in Unix *tutto è un file*, cioè può essere "aperto", "chiuso", "letto", "scritto" eccetera; questi "file speciali" possono essere delle comunicazioni fra processi, delle pipe, delle interfacce hardware o altro ancora: il sistema operativo si occuperà di gestire tutto nel modo appropriato lasciando ai programmi l'illusione di stare usando un normale file.

Poiché accedere ad un archivio informatico da una memoria di massa è una operazione piuttosto lenta, necessita di una conoscenza dettagliata dell'hardware del computer e crea un rischio di conflitto di risorse fra programmi in esecuzione, queste operazioni sono eseguite dal sistema operativo per conto dei programmi che le richiedono. **Le operazioni più comuni, universalmente presenti in tutti i sistemi operativi, sono:**

- **Apertura:** il programma segnala al sistema operativo che ha necessità di accedere a un certo file. Il sistema operativo controlla che il file esista e che non sia già usato o bloccato, da esso o da un altro programma; crea alcune strutture dati per gestire le operazioni successive e riserva una certa quantità di memoria RAM, detta buffer, per memorizzare i dati in transito da e per il file; restituisce al programma un simbolo (o un *handle* oppure un indirizzo di memoria fisica) a cui il programma farà riferimento nelle successive operazioni sul file; gli altri programmi possono accedere allo stesso file solo in modo limitato (solo lettura) o non possono accedervi affatto.
- **Lettura:** il programma richiede dei dati dal file. Il sistema operativo li legge, li memorizza nel buffer di cui comunica l'ubicazione al programma richiedente.
- **Scrittura:** Il programma vuole scrivere dei dati in un file. Per prima cosa memorizza i dati in questione in un buffer, di cui poi comunica l'ubicazione al sistema operativo.
- **Chiusura:** il programma comunica che non ha più bisogno del file. Tutte le strutture dati e il buffer allocato vengono rilasciate; il sistema operativo elimina il file dalla lista di quelli in uso; gli altri programmi hanno ora libero accesso al file.

Di per sé il sistema operativo è in grado di compiere le seguenti operazioni:

- **Riallocazione:** il sistema operativo (file system) è in grado di spostare o muovere il file da una directory all'altra tramite ad esempio copia e incolla o da terminale a riga di comando.
- **Rinominazione:** il sistema operativo è in grado di rinominare il file assegnandogli un nome diverso per via grafica o da riga di comando.
- **Eliminazione:** il sistema operativo è in grado di rimuovere, se richiesto dall'utente, il file dalla memoria di massa che lo contiene, andando però solo ad eliminare il collegamento logico-fisico del file in memoria, il cui spazio di memoria risulterà dunque non vuoto, ma *libero* nel senso di sovrascrivibile.

Un programma non può leggere o scrivere un file se prima non lo ha "aperto", e una volta finito di usarlo lo deve sempre "chiudere" per dare modo al sistema operativo di liberare le risorse occupate.

Apertura file in php

Quando si apre un file, bisogna specificare le seguenti informazioni:

- Se lo si vuole aprire in sola lettura, in sola scrittura, o in lettura e scrittura
- Se si vuole che il puntatore sia posizionato all'inizio o alla fine del file
- Cosa fare se il file non esiste

Modalità di apertura dei file di testo

In PHP, un file si apre con il comando `fopen(filename, mode)`, dove il mode può essere uno dei seguenti:

Mode	Descrizione
'r'	Solo lettura, puntatore all'inizio del file
'r+'	Lettura e scrittura, puntatore all'inizio del file
'w'	Solo scrittura, puntatore all'inizio del file. Se il file non esiste, tenta di crearlo
'w+'	Lettura e scrittura, puntatore all'inizio del file. Se il file non esiste, tenta di crearlo
'a'	Solo scrittura, puntatore alla fine del file. Se il file non esiste, tenta di crearlo
'a+'	Lettura e scrittura, puntatore alla fine del file. Se il file non esiste, tenta di crearlo

Lettura da file in php

In PHP, un file si legge con il comando `fread(risorsa, lunghezza)`, dove la lunghezza dice fino a che punto si vuole leggere il file.

Per esempio:

```
<?php
$nomefile = "..\Esempi-XML\sms15.xml";
$handle = fopen ($nomefile, "r");
$contenuto = fread ($handle, filesize $nomefile);
fclose ($handle);
?>
```

Scrittura su file in php

In PHP, un file si scrive con il comando `fwrite(risorsa, stringa)`.

Esempio di scrittura in cima al file:

```
<?php
$nomefile="gatta.txt";
$testo="Tanto va la gatta al lardo che ci lascia lo zampino";
$handle = fopen($nomefile, 'w');
fwrite($handle, $testo);
fclose($handle);
?>
fwrite($handle, "<HTML>\n");
fwrite($handle, "<BODY>\n");
fwrite($handle, "<TABLE>\n");
fwrite($handle, "<TR>\n");
fwrite($handle, "<TD>Nome</TD><TD>$_POST[nome]</TD>\n");
fwrite($handle, "</TR>\n");
fwrite($handle, "</TABLE>\n");
fwrite($handle, "</BODY>");
fwrite($handle, "</HTML>");
```

Scrittura su file – 2 in php

Esempio di scrittura in fondo al file (append):

```
<?php
$nomefile="gatta.txt";
$testo="Tanto va la gatta al lardo che ci lascia lo zampino";
$handle = fopen($nomefile, 'a');
fwrite($handle, $testo);
fclose($handle);
?>
```

La gestione completa del file di testo usando i vettori per la modifica e la cancellazione senza usare un secondo file

Ipotisi interessante e fattibile ma pericolosa. Se nel trasferimento dei dati in memoria e l'azzeramento del file dovesse andare via la corrente si perderebbe tutto. Questo sistema è comunque possibile utilizzarlo, vediamo come:

I **file testuali** sono dei semplici documenti di testo (**.txt**) in cui vengono salvati dei dati per poi essere letti tramite script php.

In particolare php offre una gamma di funzioni atte alla scrittura e lettura dei file in cui potranno essere salvati dei dati e letti all'occorrenza.

LIMITI Degli archivi TESTUALI

Occorre fin da subito chiarire un punto: i database testuali sono utilizzabili solo nel caso in cui **la mole di dati in esso contenuta è relativamente piccola e soggetti a sporadiche modifiche**.

Infatti essi avranno una gestibilità decisamente molto più complessa e limitata rispetto a quella ottenibile con un database MySQL dato che non sarà possibile formulare le query.

Inoltre, risulta quasi del tutto impossibile istaurare relazioni fra i dati.

IL FUNZIONAMENTO

Al fine di gestire i dati contenuti in un file .txt occorrerà:

- "scrivere" il file con una idonea formattazione;
- eseguire operazioni su di esso ricorrendo ad alcune funzioni; fra le tante, ve ne sono alcune che assumono particolare rilievo: fopen(), fread(), fwrite(), fclose(), file() ed explode().

Chiariamo i due concetti esaminandoli separatamente.

LA FORMATTAZIONE DI UN file TESTUALE

Un file .txt si articola in **più righe**: ognuno di essi potrà rappresentare un **set di dati**.

Ad esempio poniamo di voler costruire un database in cui salvare i dati riguardanti dei prodotti su un file di testo: ogni rigo potrà rappresentare un prodotto.

Tuttavia, per ogni prodotto inserito occorrerà salvare più di una informazione: poniamo, ad esempio, di voler salvare **il nome, la taglia e il prezzo**.

Affinche più dati siano scritti su un unico rigo occorrerà strutturare una formattazione adeguata. Vediamone un esempio:

capo|taglia|prezzo

Pertanto il file di testo che possiamo chiamare **prodotti.txt** avrà una struttura di questo genere:

1. camicia|S|150
2. felpa|XXL|220
3. t-shirt|M|250

In pratica i tre dati (nome, taglia e prezzo) che desideriamo salvare su un'unica riga del file .txt saranno separati gli uni dagli altri da un **carattere separatore** (poco comune) a nostra scelta, nell'esempio "|".

Quindi, **un file testuale è un file .txt in cui ciascun rigo contiene uno o più dati e all'interno del rigo i singoli dati sono separati da un carattere separatore.**

Ciò sarà rilevante sia in fase di lettura, sia in fase di scrittura del file.

LA LETTURA

In fase di "lettura" è possibile far ricorso a diverse funzioni che tuttavia presentano ognuna delle specifiche particolarità.

Quella che in caso di database testuali risulta essere maggiormente utile è file().

La funzione file() accetta come unico parametro il percorso al file che si desidera leggere ed esegue una **lettura del file rigo per rigo restituendo un array in cui ciascun elemento è costituito da un rigo.**

Ovviamente, avendo ottenuto un array, questo potrà essere letto con un classico ciclo foreach.

Scorrendo il nostro array otterremo un singolo rigo per ogni ciclo; per isolare i singoli dati presenti all'interno di tale rigo si farà ricorso alla funzione explode().

La funzione explode() riceve due parametri obbligatori: il primo è il carattere separatore; il secondo è una stringa da trasformare in un array.

Vediamo cosa avremo:

```

1. <?php
2. $my_database_txt = 'prodotti.txt';
3. $array_righi = file($my_database_txt);
4. foreach($array_righi as $key => $capi){
5.     list($capo, $taglia, $prezzo) = explode("|", $capi);
6.     echo '
7.     <p>
8.     Capo: '$capo.' <br />
9.     Taglia: '$taglia.' <br />
10.    Prezzo: '$prezzo.' <br />
11.    <a href="action.php?delete='.$key.'">Elimina</a> -
    <a href="form_update.php?row='.$key.'">Modifica</a>
12.    </p>
```

```

13.     <hr />;
14.   }
15. ?>

```

Si ponga attenzione ai link per eseguire l'eliminazione e la modifica di un prodotto il cui funzionamento sarà di seguito illustrato.

LA SCRITTURA

Per poter scrivere su un file di testo occorre anzitutto assicurarsi che questo abbia i permessi di scrittura: a questo scopo può essere utile la funzione `is_writable()`. Essa prende come parametro il percorso al file e restituisce un valore booleano, TRUE se il file può essere scritto.

La scrittura del file avverrà con le funzioni `fopen()`, `fwrite()` e `fclose()`.

La funzione `fopen()` serve per aprire il collegamento con la risorsa (il file da scrivere). Essa prevede due parametri obbligatori: il percorso al file e una stringa che ci indicherà modalità con la quale si vorrà operare sul file (si rimanda al manuale per maggiori dettagli).

La funzione `fwrite()` esegue la scrittura sul file e prevede due parametri: la risorsa e la stringa da scrivere. Infine, la funzione `fclose()` esegue la chiusura del file e prevede come unico parametro la risorsa.

Avremo una pagina con un banale form:

```

1.  <form action="action.php" method="post">
2.  <label for="capo">Capo</label>
3.  <input type="text" id="capo" name="capo" />
4.  <label for="taglia">Taglia</label>
5.  <input type="text" id="taglia" name="taglia" />
6.  <label for="prezzo">Prezzo</label>
7.  <input type="text" id="prezzo" name="prezzo" />
8.
9.  <input type="submit" name="scrivi" value="scrivi" />
10. </form>

```

E poi avremo una pagina (action.php) che eseguirà la scrittura sul un file prodotti.txt:

```

1.  <?php
2.  $my_database_txt = 'prodotti.txt';
3.  if(isset($_POST['scrivi']))
4.  {
5.      if(!is_writable($my_database_txt)){
6.          exit("il file non ha i permessi di scrittura!");
7.      }
8.      // riceviamo i dati e li filtriamo
9.      $bad_char = array("|", "rn", "r", "n");
10.     $capo = str_replace($bad_char, "", $_POST['capo']);
11.     $taglia = str_replace($bad_char, "", $_POST['taglia']);
12.     $prezzo = str_replace($bad_char, "", $_POST['prezzo']);
13.     // apriamo il file
14.     $open = fopen($my_database_txt, "a+");
15.     // scriviamo i dati separati dal carattere separatore
16.     fwrite($open, $capo."|".$taglia."|".$prezzo."rn");
17.     // chiudiamo il file
18.     fclose($open);
19.
20.     // ritorniamo nella pagina di visualizzazione

```

```

21. header("location: lettura.php");
22. exit;
23. }
24. ?>

```

ELIMINARE UN RIGO DAL FILE .TXT

Eliminare un rigo sta a significare, nel caso del nostro database basato su file di testo, eliminare un prodotto. Abbiamo precedentemente visto che ciascun prodotto avrà un link per l'eliminazione scritto attraverso un ciclo foreach di questo tipo (si veda il codice per eseguire la lettura):

```

1. <a href="action.php?delete=' . $key. '">Elimina</a>

```

La variabile \$key all'interno del ciclo individua la chiave dell'array ottenuto con la funzione file() e, quindi, ci consente di individuare il rigo specifico che vogliamo eliminare. Nel file action.php pertanto avremo:

[view plaincopy to clipboardprint?](#)

```

1. if(isset($_GET['delete']))
2. {
3.     // creiamo l'array con tutti i rigi
4.     $array_righi = file($my_database_txt);
5.     // eliminiamo dall'array il rigo la chiave inviata via get
6.     unset($array_righi[$_GET['delete']]);
7.     // apriamo il file resettando il contenuto
8.     $open = fopen($my_database_txt, "w");
9.     foreach($array_righi as $key => $value){
10.        // ri-scriviamo tutti i rigi (rimanenti)
11.        fwrite($open, $value);
12.    }
13.    fclose($open);
14.    // ritorniamo nella pagina di visualizzazione
15.    header("location: lettura.php");
16.    exit;
17. }

```

Occorre notare che con la funzione unset() eliminiamo l'elemento dall'array; inoltre la funzione fopen, a differenza di quando fatto in precedenza, avrà come secondo parametro w e quindi questo verrà cancellato di tutto il suo contenuto e riscritto di nuovo ma senza il rigo eliminato.

Il codice proposto necessiterebbe di ulteriori e più rigidi controlli (quantomeno con array_key_exists) ma ho voluto ridurre al minimo lo script.

LA MODIFICA DI UN RIGO DI UN FILE .TXT

La modifica del singolo rigo segue una logica per molti aspetti simile a quella vista per l'eliminazione. Tuttavia in questo caso avremo uno step ulteriore: dovremo precompilare un form con i dati correnti del rigo che desideriamo modificare e poi, al submit di tale form, eseguire la modifica.

Anzitutto esaminiamo il link che rimanda alla pagina di modifica che, ricordo, viene prodotto all'interno del ciclo foreach (si veda il codice per eseguire la lettura):


```
1. <a href="form_update.php?row=' . $key. '">Modifica</a>
```

Come detto in precedenza per l'eliminazione, \$key ci permetterà di individuare il rigo che desideriamo modificare. In questo caso dovremo estrarre i dati contenuti in tale rigo e compilare i value del form:

```
1. <?php
2. $my_database_txt = 'prodotti.txt';
3. if(!isset($_GET['row'])){
4.     header("location: lettura.php");
5.     exit;
6. }
7. $array_righi = file($my_database_txt);
8. if(!isset($array_righi[$_GET['row']])){
9.     exit('errore nella chiave');
10. }
11. list($capo, $taglia, $prezzo) = explode("|", $array_righi[$_GET['row']]);
12. ?>
13. <html>
14. <head>
15. </head>
16. <body>
17. <h2><a href="lettura.php">Torna alla lista degli articoli</a></h2>
18. <form action="action.php" method="post">
19. <label for="capo">Capo</label>
20. <input type="text" id="capo" name="capo" value="<?php echo htmlentities($capo, ENT_QUOTES); ?>" />
21. <label for="taglia">Taglia</label>
22. <input type="text" id="taglia" name="taglia" value="<?php echo htmlentities($taglia, ENT_QUOTES); ?>" />
23. <label for="prezzo">Prezzo</label>
24. <input type="text" id="prezzo" name="prezzo" value="<?php echo htmlentities($prezzo, ENT_QUOTES); ?>" />
25. <input type="hidden" name="row_update" value="<?php echo $_GET['row']; ?>" />
26. <input type="submit" name="modifica" value="modifica" />
27. </form>
28. </body>
29. </html>
```

Si noti il campo **name="row_update"** di tipo **hidden** che avrà come value la chiave inviata via get. All'invio del form andremo a recuperare i dati inviati: i nuovi valori che dovrà assumere il rigo e il rigo da modificare (presente nel campo hidden). Pertanto avremo:

```
1. $my_database_txt = 'prodotti.txt';
2. if(isset($_POST['modifica']) AND isset($_POST['row_update']))
3. {
4.     // creiamo l'array con tutti i righi
5.     $array_righi = file($my_database_txt);
6.     // riceviamo i dati e li filtriamo
7.     $bad_char = array("|", "rn", "r", "n");
8.     $capo = str_replace($bad_char, "", $_POST['capo']);
9.     $taglia = str_replace($bad_char, "", $_POST['taglia']);
10.    $prezzo = str_replace($bad_char, "", $_POST['prezzo']);
11.    // ri-scriviamo il rigo (che sostituirà il precedente)
12.    $array_righi[$_POST['row_update']] = $capo."|".$taglia."|".$prezzo."rn";
13.    // apriamo il file resettando il contenuto
14.    $open = fopen($my_database_txt, "w");
15.    foreach($array_righi as $key => $value){
16.        // ri-scriviamo tutti i righi
17.        fwrite($open, $value);
18.    }
19.    fclose($open);
20.    // ritorniamo nella pagina di visualizzazione
```

```
21. header("location: lettura.php");  
22. exit;  
23. }
```

Come possiamo notare lo script segue una logica analoga a quella vista per l'eliminazione con l'unica differenza che anzich  eliminare l'elemento dall'array \$array_righi con unset lo andremo a valorizzare con i nuovi valori (opportunamente filtrati) provenienti dal form, separando i valori con il carattere separatore | e aggiungendo a fine stringa il ritorno a capo.

L'aggiornamento e la cancellazione di un record da un file di testo usando due archivi txt (metodo sicuro senza il rischio di perdere i dati)

Rispetto al metodo precedente, in questo utilizziamo due file, il primo che viene utilizzato per la memorizzazione dei record in coda, la fase di inserimento viene realizzata senza problemi. Nella fase di modifica creeremo un file di appoggio che servirà a contenere il contenuto del "vecchio" file con l'aggiornamento del record e successivamente si cancella il vecchio e si rinomina il nuovo con il vecchio nome del file, in questo modo avremo correttamente aggiornati i record del file. La procedura per la cancellazione usa, in linea di massima, lo stesso sistema con qualche accorgimento in più. (Le pagine Html non sono state inserite nell'esempio per l'ovvietà dell'esempio, a noi interessa studiare le pagine Php).

Per la modifica in Php, si procede con il ciclo for prendendo i dati che vogliamo modificare, il ciclo si avvia con il contatore da 1, `$i=1` fino a `$i<$num` che in questo caso è 5. Preleviamo dal form il record per modificare e procediamo all'apertura del file di testo con `fopen` in lettura. Successivamente creiamo con la funzione `file` un array con ogni singolo record del file di testo su cui vogliamo attuare la modifica, blocchiamo il tutto con `flock` e inizializziamo la variabile `$trovato` a 0.

Apriamo poi il secondo file di testo in scrittura per modificare il tutto con l'opportuno ciclo che trascrive gli elementi del vettore che vogliamo modificare; elementi che erano presenti già nel primo file di testo e procediamo con la fase di scrittura (`fwrite`). Con la funzione `implode` trasformeremo il contenuto del vettore in opportune stringhe separate e chiudiamo il contenuto con `flock` e `fclose`. Abbiamo ottenuto così la modifica dell'evento.

Per la cancellazione in Php, iniziamo con il recupero dei dati dal form con `$_POST` del codice inserito nella pagina html dal personale che vuole eliminare un evento. Successivamente controlliamo con un `if` se il file esiste quindi faremo: `if(file_exists($nomefile))` e se quest'ultimo esiste allora procederemo ad aprire il file in lettura (`read`) con la funzione `fopen`.

Quindi, dopo aver aperto il file in lettura, procediamo a creare un ulteriore file detto "temporaneo" che sarà inserito nella variabile `$fp_o2`, il file lo chiamiamo "temp.txt".

Blocchiamo il contenuto di `$fp_o2` con `flock` e diamo vita ad un ciclo "for" che controlli il contenuto di `$fp` che è il file in scrittura che abbiamo aperto all'inizio. Se il valore inserito non è uguale a quello presente nel file di testo allora utilizzeremo la funzione `fwrite`; nel caso contrario vuol dire che il codice è uguale alla stringa di corrispondenza. A questo punto con un `echo` comunicheremo a video la cancellazione dell'evento dalla lista e sblocchiamo il file normale e quello temporaneo e li chiudiamo con la funzione `fclose`. Infine dato che il codice non corrisponde alla stringa all'interno del vettore allora possiamo procedere alla cancellazione del vecchio file di testo con la funzione `unlink($nomefile)` e rinomineremo con la funzione `rename` il file temporaneo che diventerà a tutti gli effetti il nostro nuovo file con il contenuto adeguato.

Ecco le tre pagine in Php (inserimento, modifica e cancellazione)

Inserimento_evento.php

```
<html>
<head>
</head>
<body>
<?php echo"<body bgcolor='cococo'>";
$codice=$_POST['codice'];
$ospiti=$_POST['ospiti'];
$data=$_POST['data'];
$ora=$_POST['ora'];
$descrizione=$_POST['descrizione'];
if ($codice=="" || $ospiti=="" || $data=="" || $ora=="" || $descrizione=="")
    echo"<h1>Errore!</h1><h3> Non sono stati inseriti tutti i campi!</h3>";
    else
    {
        $g="$data[0]$data[1]";
        $m="$data[3]$data[4]";
        $a="$data[6]$data[7]";
        if(checkdate($m,$g,$a)==true)
        {
            echo"<b>Ecco il riepilogo dei dati inseriti:</b><br>
            Codice: $codice<br>
            Ospiti: $ospiti<br>
            Data: $data<br>
            Ora: $ora<br>
            Descrizione: $descrizione<br>";
            $fp=fopen("eventi.txt","a");

            if($fp)
            {
                flock($fp,2);
                $nl=chr(13).chr(10);
                fputs ($fp, "$codice,$ospiti,$data,$ora,$descrizione$nl");
                echo"<b>I dati sono stati salvati correttamente!</b><br>";
                flock($fp,3);
            }
            else echo"Non &egrave; stato possibile memorizzare i dati. Il file non esiste<br>";
                }
                else
                echo"Data inserita non corretta!<br>";
                }
            echo"Per effettuare un altro inserimento <a
href='inserimento_evento.html'>clicca qui</a>";
?>
<br>
</body>
</html>
```

Modifica_evento.php

```
<?php
$num=5; //numero dei campi del record da trasferire nel vettore da 0 a 8.
$nomefile="eventi.txt"; //variabile che contiene il nome del file prodotti
$vec_cod=$_POST['vc'];
for($i=0;$i<$num;$i++) //ciclo che recupera i dati forniti dal form per l'inserimento dei nuovi alimenti
{
    $prodotto[$i]=$_POST["p$i"];
}
```

```

}
if(file_exists($nomefile))
{
    //se il file esiste
    $fp=fopen($nomefile,"r");
    if(!$fp)
        die("<h2> Il file $nomefile non &egrave; stato aperto."); //se non va a buon fine ciò che è scritto da qui in
    avanti non viene letto
    flock($fp,2);//per operazioni di scrittura esclusiva
    $nomefile_o2="temp.txt";//file non esistente per cui verrà creato con fopen
    $fp_o2=fopen($nomefile_o2,"w");
    if(!$fp_o2)
        die("<h2> Il file $nomefile_o2 non &egrave; stato aperto.");
    flock($fp_o2,2);
    $trovato=0;
    while(!feof($fp))
    {
        $stringa=fgets($fp);
        $vettore=explode(",",$stringa);
        if($vettore[0]!=$vec_cod)
            fwrite($fp_o2,$stringa);
        else
        {
            $trovato=1;
            $nl=chr(13).chr(10);
            for($j=0;$j<count($vettore);$j++)
            {
                if($prodotto[$j])
                    $vettore[$j]=$prodotto[$j];
            }
            $stringa=implode(",",$vettore);
            if($prodotto[$j])
                fwrite($fp_o2,$stringa.$nl);
            else
                fwrite($fp_o2,$stringa.$nl);
        }
    }
}
if($trovato==0)
    die("<h3> Il codice inserito non appartiene a nessuno evento in programma.");
echo"L'evento $prodotto[0] &egrave; stato modificato con successo!";
flock($fp,3);
flock($fp_o2,3);
fclose($fp);
fclose($fp_o2);
unlink($nomefile);//eliminazione del file di testo
rename($nomefile_o2,$nomefile);//rinomino il nuovo file con il vecchio nome
}
else
    echo"Il file non esiste";
?>

```

Canc_evento.php

```

<?php
$codice=$_POST['cod']; //recupero dati da form
$nomefile="eventi.txt";
if(file_exists($nomefile))
{
    //se il file esiste

```

```
$fp=fopen($nomefile,"r");
if(!$fp)
die("<h2> Il file $nomefile non &egrave; stato aperto.");
//se non va a buon fine ciò che è scritto da qui in avanti non viene letto
flock($fp,2);//per operazioni di scrittura esclusiva
$nomefile_o2="temp.txt";//file non esistente per cui verrà creato con fopen
$fp_o2=fopen($nomefile_o2,"w");
if(!$fp_o2)
die("<h2> Il file $nomefile_o2 non &egrave; stato aperto.");
flock($fp_o2,2);
$trovato=0;
while(!feof($fp))
{
$stringa=fgets($fp);
$vettore=explode(",",$stringa);
if($vettore[0]=$codice)
fwrite($fp_o2,$stringa);
else
$trovato=1;
}
if($trovato==0)
die("<h3> Il codice inserito non appartiene a nessun evento in programma.");
echo"L'evento &egrave; eliminato dalla lista";
flock($fp,3);
flock($fp_o2,3);
fclose($fp);
fclose($fp_o2);
unlink($nomefile); //eliminazione del file di testo
rename
($nomefile_o2,$nomefile);
}
else
echo"Il file non esiste";
echo"<a href='gestione.html'>Torna all'area personale</a>". "<br><br>";
?>
```
