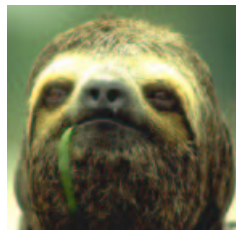

Programmare in PHP

Guida pratica alla programmazione PHP in ambiente GNU/Linux

Gianluca Giusti brdp @ urcanet.it

2003.01.20



Gianluca Giusti si è diplomato in Informatica presso l'ITIS de L'Aquila, attualmente frequenta il corso di laurea in Fisica presso la Facoltà di Scienze MM.NN.FF. dell'Università de L'Aquila

Lavora come analista programmatore nella città di Roma allo sviluppo di portali, siti di commercio elettronico, software di BI.

È appassionato di amministrazione di sistema, in particolare GNU/Linux, e di reti informatiche.

Programmare in PHP

Copyright © 2001-2002 Gianluca Giusti

brdp @ urcanet.it

This information is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This work is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this work; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Una copia della licenza GNU General Public License, versione 2, si trova nell'appendice A.

non modificare

;)

L'indirizzo della versione aggiornata dell'opera "Programmare in PHP" è:

<http://www.urcanet.it/brdp/php_manual/>

Al momento l'opera è incompleta, tuttavia ho ritenuto opportuno pubblicare quanto fatto. Scrivo questo documento nel tempo libero e ultimamente ne ho davvero poco quindi aggiornerò il contenuto ogni volta che un argomento sarà completato.

Gli script di esempio sono contenuti e raggruppati per capitolo all'indirizzo <http://www.urcanet.it/brdp/php_manual/esempi/>

L'opera è inclusa in "Appunti di informatica libera" di Daniele Giacomini.

La diffusione di questo documento è incoraggiata in base ai termini della licenza.

Indice generale

Prefazione	7
1 Avere le idee chiare	8
1.1 Cos'è il PHP?	8
1.2 Cosa può fare il PHP?	11
1.3 PHP su piattaforma Microsoft	11
1.4 Quanto costa il PHP?	11
2 Le basi del linguaggio	12
2.1 La sintassi	12
2.2 Primo script	14
2.3 Tipi di dati	15
2.3.1 integer	16
2.3.2 boolean	17
2.3.3 double	17
2.3.4 string	18
2.3.5 array	19
2.3.6 object	23
2.4 Operatori	23
2.4.1 Operatori aritmetici	23
2.4.2 Operatori di assegnazione	24
2.4.3 Operatori di controllo degli errori	25
2.4.4 Operatori di incremento e decremento	26
2.4.5 Operatori logici	27
2.4.6 Operatori di confronto	27
2.5 Strutture di controllo	27
2.5.1 if	28
2.5.2 else	29
2.5.3 elseif	30
2.5.4 switch	32
2.5.5 while	33
2.5.6 do..while	35
2.5.7 for	35
2.5.8 foreach	36
2.5.9 break	38
2.5.10 include()	39
3 Passaggio di variabili tramite l'HTTP	41
3.1 Metodo get	41
3.2 Metodo post	43
3.3 Quale metodo scegliere?	43
3.4 Cosa è cambiato nel PHP versione 4.2	44

4	L'utilizzo delle funzioni in PHP	45
4.1	Le funzioni predefinite	45
4.2	Le funzioni definite dall'utente	47
4.3	Le funzioni ricorsive	52
5	La gestione del file system	54
5.1	Concetto di file	54
5.2	Apertura di un file	54
5.3	Lettura di un file di testo	55
5.4	Scrittura in un file di testo	57
5.5	Accodare testo in un file	60
5.6	Conclusioni	62
6	Classi e oggetti	63
6.1	Definizione di classi e oggetti	63
6.2	Utilizzo di classi e oggetti	65
6.3	Esempi applicativi	67
6.4	Estensione di una classe	70
7	Nelle prossime puntate	71
	Appendice A GNU GENERAL PUBLIC LICENSE	74
	Indice analitico	81

Prefazione

Questo documento vuole essere una guida all'utilizzo di uno dei più versatili e potenti linguaggi di programmazione oggi diffuso in Internet. Il lettore non troverà un elenco delle funzioni del linguaggio, perché sarebbe inutile dal momento che queste notizie sono già disponibili nel manuale ufficiale; al contrario, potrà trovare una guida all'utilizzo degli strumenti che gli sviluppatori del linguaggio hanno messo a disposizione.

Naturalmente ognuno potrà ampliare e personalizzare gli script proposti come esempio; sarà la guida stessa a sollevare dubbi e a suggerire delle modifiche, per il resto via libera all'immaginazione del lettore. Inutile dire che per qualsiasi problema la mia casella di posta è a vostra disposizione compatibilmente con gli impegni.

Per la lettura della guida si darà per scontato:

- la configurazione corretta della macchina che offre i servizi, in cui devono essere installati il server HTTP (negli esempi di questa guida si fa riferimento principalmente ad Apache), il modulo PHP 4 e una base di dati (DBMS), che negli esempi proposti è costituita da MySQL;
- la conoscenza del linguaggio HTML;
- un minimo di conoscenza del linguaggio SQL per l'interrogazione delle basi di dati di prova.

Verranno risolti i problemi più comuni, progettati e realizzati i servizi necessari a un sito per essere moderno, dinamico e al passo coi tempi. Si pensi alla gestione di utenze, aree riservate, notizie, sondaggi, motori di ricerca, album fotografici, servizi FTP, ecc.

Per fare questo si partirà da esempi molto semplici, scritti in modo altrettanto semplice, fino ad arrivare alla realizzazione di «classi» da utilizzare in tutti i nostri servizi.

Prima di iniziare converrà collegarsi al sito [<http://www.php.net>](http://www.php.net) per scaricare il manuale ufficiale del PHP; sarà molto utile ed è importante saperlo consultare. Viene rilasciato in vari formati; probabilmente il più conveniente è il formato PDF. In tal modo, si disporrà di un unico file su cui poter effettuare ricerche di testo. Inoltre è importante ricordare che un gruppo di persone sta traducendo il manuale in italiano ospitato sul sito ufficiale all'indirizzo [<http://www.php.net/manual/it/>](http://www.php.net/manual/it/). Chi volesse partecipare può mettersi in contatto con i coordinatori del progetto Luca Perugini Simone Cortesi.

Avere le idee chiare

Prima di addentrarsi nella programmazione, è bene capire quali sono gli strumenti a disposizione e cosa permettono di fare. C'è molto interesse verso le tecnologie utilizzate in Internet, di conseguenza si crea spesso confusione. Una delle conseguenze più frequenti è ostinarsi a utilizzare un linguaggio nato con uno scopo per realizzare tutt'altro.

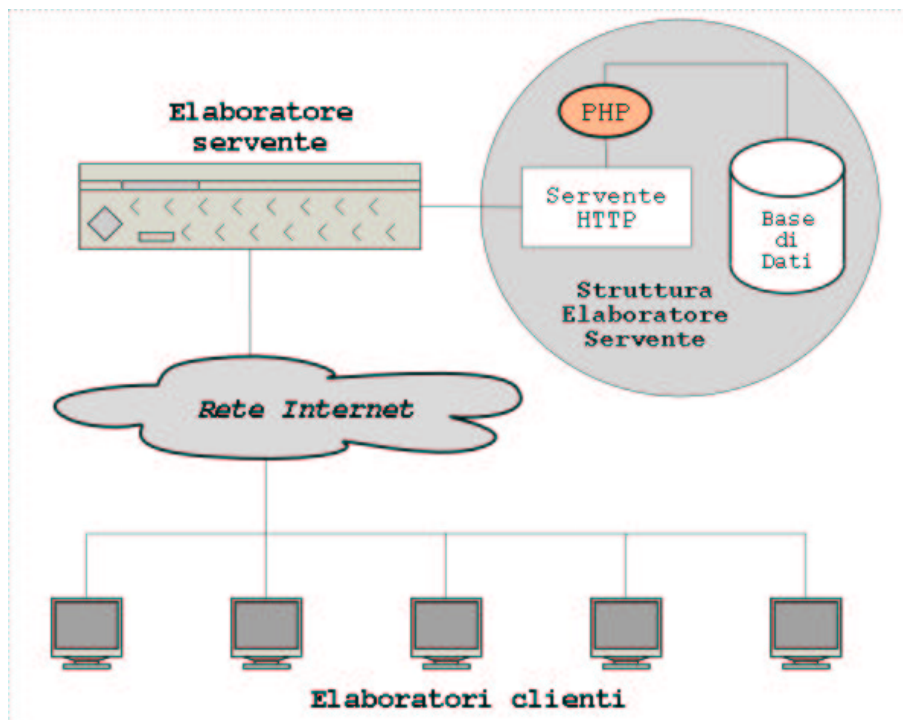
1.1 Cos'è il PHP?

PHP sta per *Hypertext Preprocessor*. Dalla documentazione ufficiale il PHP viene definito come un «linguaggio script dal lato del server immerso nel codice HTML». Di seguito la definizione nel dettaglio:

«linguaggio script»: il PHP è un vero e proprio linguaggio di programmazione, è importante rendersi conto che l'HTML, ad esempio, non è un linguaggio di programmazione ma un linguaggio di descrizione e formattazione del testo. Inoltre, per i più esperti, il PHP è un linguaggio interpretato.

«Dal lato del server»: queste le parole fondamentali. Il codice PHP inserito tra i marcatori HTML viene interpretato dall'elaboratore server e non dal navigatore del cliente. Il modo migliore per rendere l'idea è passare ad un esempio.

Figura 1.1. Il server elabora tramite l'interprete PHP gli script i quali generano le pagine HTML che vengono fornite al cliente tramite il protocollo HTTP.



Si supponga di voler scrivere la data odierna in una pagina HTML. L'HTML non permette di farlo, in quanto, è un linguaggio di formattazione statico. Una soluzione potrebbe essere l'utilizzo di uno script Java di Netscape (JavaScript) ma è un linguaggio lato cliente. Il codice JavaScript viene eseguito dal navigatore del visitatore e non dal server. Tutto ciò potrebbe portare alla visualizzazione di una data errata sulle pagine del sito solo perché l'orario di sistema del visitatore non è impostato correttamente. Inoltre la guerra per il monopolio nel mercato dei navigatori ipertestuali tra Microsoft e Netscape ha fatto sì che non ci sia uno standard, di conseguenza Netscape non

gestisce molte istruzioni JScript di Microsoft e Explorer non riconosce molte istruzioni JavaScript di Netscape ¹.

A questo punto l'ideale è fornire al navigatore una stringa HTML generata e formattata da una funzione PHP. Ecco come:

```
1 <html>
2   <head>
3     <title>La mia prima pagina php</title>
4   </head>
5   <body>
6
7     <br><br>
8
9     <?php echo date("d-m-Y") ?>
10
11    <br><br>
12
13    <a href="../cap_2/benvenuto.php">vai a benvenuto.php &gt;&gt;</a>
14
15  </body>
16 </html>
```

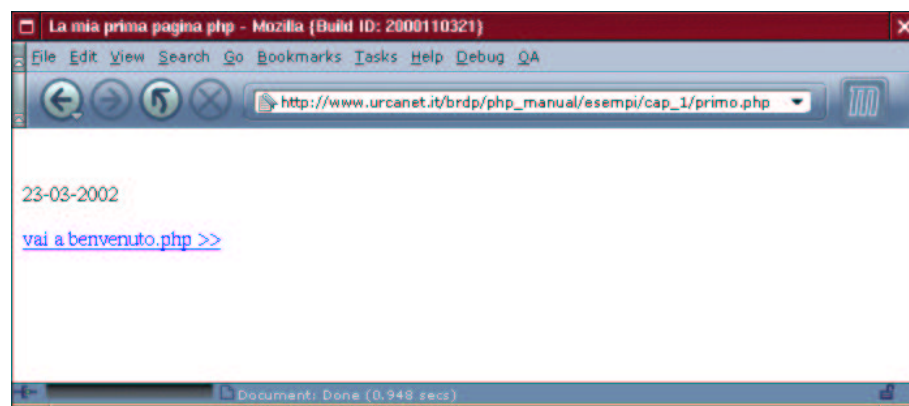
Una volta completato salvare il codice nel file 'primo.php'.

Se si prova ad aprire 'primo.php' dal menù *file* del navigatore il risultato è una pagina bianca. Il motivo sta nel fatto che il PHP è un «linguaggio dal lato del servente» dunque il navigatore (cliente) non è in grado di interpretarne il codice compreso tra i marcatori: '<?php ?>'

Per visualizzare la data in modo corretto è necessario dare il codice in pasto all'interprete PHP tramite il servente HTTP, quindi bisogna copiare il file 'primo.php' nella directory dei documenti del servizio HTTP e dal navigatore richiamare il file scrivendo l'indirizzo dell'elaboratore su cui è ospitato. Tutti gli script di esempio sono disponibili in rete all'indirizzo <http://www.urcanet.it/brdp/php_manual/esempi/> e sono raccolti per capitolo.

Se si scaricano gli script di esempio per essere eseguiti in locale e la configurazione del servente è corretta, il risultato sarà:

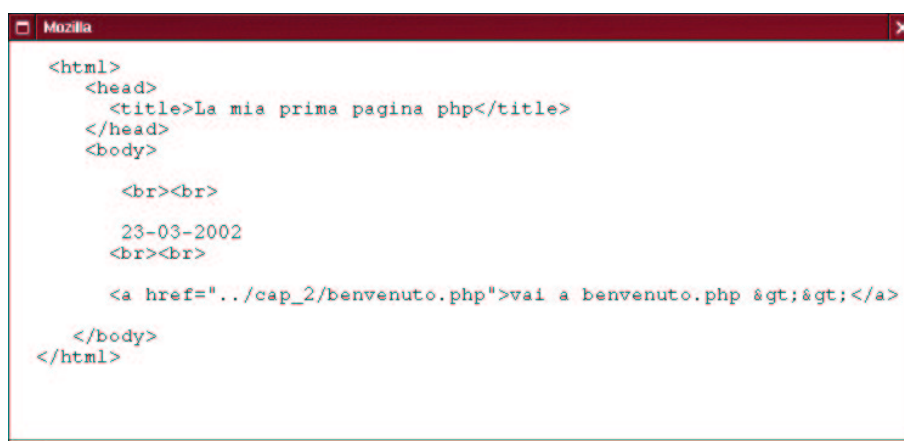
Figura 1.2. Ecco come appare la pagina HTML <http://www.urcanet.it/brdp/php_manual/esempi/cap_1/primo.php> sul navigatore del visitatore.



Ben più interessante è osservare il sorgente della pagina <http://www.urcanet.it/brdp/php_manual/esempi/cap_1/primo.php> dal menu *visualizza* del navigatore, ecco quello che viene inviato dal servente:

¹Netscape ha sviluppato un linguaggio di programmazione ad oggetti interno al navigatore chiamato JavaScript tramite il quale è possibile gestire le finestre del navigatore, i form di invio dati, la navigazione stessa delle pagine e tanto ancora. La Microsoft per il proprio navigatore Explorer ha realizzato un linguaggio molto simile chiamato JScript. Purtroppo i due linguaggi non sono compatibili al 100% anche se le istruzioni di base sono ben gestite da entrambi.

Figura 1.3. L'interprete PHP ha fatto il suo dovere. Questo è il codice generato e fornito al navigatore tramite il servente HTTP.



Una volta ricevuta la richiesta della pagina da parte del navigatore, il servente inizia a leggere il file 'primo.php' e a inviarlo al cliente, appena trovati i marcatori '<?php ?>' capisce che deve farsi «tradurre» il codice in essi contenuto. Lo passa quindi all'interprete PHP che lo esegue e restituisce il risultato. In questo semplice esempio il risultato dell'operazione è la stringa '23-03-2002'.

La funzione utilizzata nell'esempio '`date("d-m-Y")`' formatta la data nello schema *giorno-mese-anno* (numerico), mentre '`echo`' visualizza a schermo il risultato della funzione '`date()`'.

Una modifica interessante al file precedente è:

```

1 <html>
2   <head>
3     <title>La mia prima pagina php</title>
4   </head>
5   <body>
6
7     <br><br>
8
9     <?php phpinfo() ?>
10
11    <br><br>
12
13    <a href='../cap_2/benvenuto.php'>vai a benvenuto.php &gt;&gt;</a>
14
15  </body>
16 </html>

```

Questa volta il risultato è una pagina che visualizza la configurazione dell'elaboratore servente, dell'interprete e dei componenti per l'interrogazione delle basi di dati, oltre a una serie di variabili predefinite. È bene dare un'occhiata al sorgente HTML della pagina, tutta opera dell'interprete PHP.

«immerso nel codice HTML»: con ciò si intende che quanto contenuto tra i marcatori '<?php ?>' viene prima interpretato, e poi inviato al navigatore come semplice codice HTML insieme al resto della pagina.

1.2 Cosa può fare il PHP?

Dunque il PHP permette di rendere dinamiche le pagine HTML dal lato del server. Questo insieme alla sua semplicità di utilizzo, alla somiglianza con la sintassi C e alle innumerevoli funzioni che permettono di interagire con basi di dati e servizi di ogni genere ne hanno decretato il successo in Internet.

La diffusione è dovuta alla semplicità di configurazione e alla completa compatibilità con numerosi server HTTP. Per il test degli script di questa guida è stato utilizzato come server Apache.²

Il PHP permette, in modo semplice e diretto, di far interagire il cliente con le basi di dati ospitate sul server tramite il semplice utilizzo di un comune navigatore.

Tutti i maggiori DBMS sono gestiti da PHP, senza dover installare software aggiuntivo o commerciale. Un breve elenco è riportato nella tabella 1.1.

Tabella 1.1. Queste le basi di dati gestite dal PHP.

BASIS DI DATI		
Adabas D	InterBase	PostgreSQL
dBase	FrontBase	Solid
Empress	mSQL	Sybase
FilePro (read-only)	Direct MS-SQL	Velocis
IBM DB2	MySQL	Unix dbm
Informix	ODBC	Ingres
Oracle (OCI7 and OCI8)		

Inoltre il PHP permette di interagire con numerosi servizi tramite i protocolli: IMAP, SNMP, NNTP, POP3, HTTP e molti altri.

1.3 PHP su piattaforma Microsoft

Il PHP è compatibile con molte piattaforme di sviluppo, tra cui quella Microsoft. Ovviamente le migliori prestazioni si ottengono in ambiente GNU/Linux (o più in generale su sistemi Unix).

Per informazioni più dettagliate si rimanda al sito ufficiale <<http://www.php.net>>.

1.4 Quanto costa il PHP?

Il PHP viene distribuito nei termini della: **The PHP License, version 2.02**.

I requisiti minimi richiesti all'elaboratore server sono del tutto abordabili. Per i test degli script contenuti in questa guida è stato utilizzato un vecchio P-120MHz con soli 16 Mibyte di RAM e un piccolo disco fisso da 1,2 Gbyte.

Su di esso è stata installata una versione Debian GNU/Linux 2.2 con server Apache, MySQL, interprete PHP4. Sulla stessa macchina erano in esecuzione il server FTP, PostgreSQL, Postfix e altri servizi minori.

²È possibile prelevare Apache dal sito ufficiale <<http://www.apache.org>>.

Le basi del linguaggio

Per poter scrivere codice in un qualunque linguaggio è necessario conoscerne la sintassi. In questo capitolo verranno trattate le espressioni fondamentali del PHP con il supporto di script di esempio.

2.1 La sintassi

In realtà ‘<?php ?>’ non sono gli unici marcatori per fornire codice all’interprete. Ecco tutti i modi di includere codice PHP negli script:

- <? *codice* ?>
- <?php *codice* ?>
- <script language="php"> *codice* </script>
- <% *codice* %>

Il primo è disponibile solo se sono stati impostati i marcatori abbreviati, ciò può essere fatto abilitando nel file di configurazione ‘php.ini’ l’opzione ‘**short_open_tag**’. Per i particolari sugli altri marcatori si rimanda al manuale ufficiale.

Va sottolineato che il codice può essere bloccato e ripreso in ogni punto dello stesso file, ad esempio:

```
1 <html>
2   <head>
3
4       <? $nome = "BRDP"; ?>
5       <title>ancora agli inizi</title>
6
7   </head>
8   <body>
9
10      <br>Ciao a tutti questo è il mio nome: <?=$nome ?>
11
12  </body>
13 </html>
```

Nell’esempio il codice viene aperto per la prima volta nella riga 4, viene assegnato un valore alla variabile ‘\$nome’, viene chiuso e poi riaperto nella riga 10 dove viene visualizzato a schermo il valore della variabile insieme a una frase HTML. Il file quindi restituirà come output la stringa:

```
Ciao a tutti questo è il mio nome: BRDP
```

Con l’occasione, nell’esempio, sono stati toccati due concetti base.

Il primo è che le variabili in PHP iniziano con il carattere ‘\$’ e sono sensibili a maiuscole e minuscole, l’altro è che non è necessario definirne il tipo, sarà quello del primo dato ad essa assegnato. Nel nostro caso la variabile ‘\$nome’, dalla riga 4 in poi, sarà considerata una stringa. Per averne conferma basta sostituire il codice della riga 4 con le due righe seguenti:

```
4 <?
5     $nome = "BRDP";
6     echo gettype($nome);
7 ?>
```

Il risultato è:

```
string
Ciao a tutti questo è il mio nome: BRDP
```

Più avanti si apprenderà come trattare i vari tipi di dati, quindi è inutile dilungarsi in questo esempio; bisogna però dire che è possibile cambiare il tipo di dati mediante l'istruzione:

```
settype($variabile, tipo_variabile);
```

e ottenere il tipo di dato contenuto in una variabile tramite:

```
gettype($variabile);
```

Il secondo concetto è nella riga 10; per visualizzare il contenuto della variabile è stata utilizzata la sintassi:

```
<?=$nome ?>
```

Per chi si avvicina al PHP dopo aver sviluppato in Perl o in altri linguaggi script CGI la differenza è lampante. Non è necessario scrivere codice per generare HTML, semplicemente si inserisce codice PHP tra i marcatori HTML. Inoltre nel caso in cui l'istruzione sia la semplice visualizzazione di una singola variabile, la sintassi classica:

```
<? echo $nome ?>
```

può essere abbreviata con:

```
<?=$nome ?>
```

Attenzione: se il codice PHP è formato da più istruzioni consecutive è richiesto il carattere punto e virgola (;) alla fine di ognuna di esse, se invece l'istruzione da inserire è una sola, il ';' può essere omissa.

Sarebbe buona norma dedicare molta cura all'indentazione del codice e inserire commenti chiari e diretti; così facendo si rende il codice più leggibile e comprensibile. La gestione dei commenti è simile a quella utilizzata in C, C++ e shell di Unix. I caratteri di commento sono: per una singola riga '/' la barra obliqua doppia oppure il carattere '#', per più righe il commento viene aperto con '/' e chiuso con '*/'. Nel secondo caso bisogna fare attenzione a non annidare i commenti. Questo è il modo corretto:

```
0    <?
1        // Questo è un commento su una singola riga
2        // adesso visualizzo ciao mondo!
3
4        echo "ciao mondo!";
5
6        /*
7        questo è un commento
8        su più righe, posso scrivere di tutto!!!
9        anche istruzioni php che non verranno interpretate come:
10       echo "questo non verrà visualizzato!";
11       chiudo il commento
12       */
13
14       echo "<br>questo invece si vedrà!!";
15
16       /*
17       Notare che ho inserito <br> un marcatore html nel
18       codice php! Posso farlo.
19       */
20    ?>
```

Questo è sbagliato:

```
0    <?
1
2        /*
3        echo "ciao mondo!"; /* questo causa problemi! */
4        */
5
6    ?>
```

2.2 Primo script

Come primo esempio verrà realizzata una pagina di riconoscimento del visitatore. Il PHP fornisce una serie di «variabili predefinite». Tra esse ce ne sono alcune fornite dal protocollo HTTP.

Per l'elenco completo delle variabili predefinite è bene consultare il manuale ufficiale del PHP, molte di esse vengono visualizzate nella pagina generata dalla funzione `'phpinfo()'`¹

Il primo passo è scegliere le variabili da utilizzare.

Per questo semplice script verranno utilizzate:

- `'SERVER_NAME'`: nome dell'elaboratore su cui risiede e viene eseguito lo script;
- `'HTTP_REFERER'`: l'URI, se ne esiste uno, da cui proviene il visitatore;²
- `'HTTP_USER_AGENT'`: il tipo di navigatore utilizzato dal visitatore;
- `'REMOTE_ADDR'`: l'indirizzo IP dell'elaboratore cliente;
- `'SCRIPT_NAME'`: il percorso dello script in esecuzione;

è importante fare molta attenzione all'utilizzo di queste variabili, alcune di esse usate in modo superficiale potrebbero fornire informazioni utili ai malintenzionati. Ad esempio `'DOCUMENT_ROOT'` fornisce la directory su file system in cui si trovano gli script PHP e i documenti del servizio HTTP. Informazioni di questo tipo è meglio tenerle riservate!

Scelte le variabili va creato il nuovo file e salvato con il nome di `'benvenuto.php'`. La prima versione, la più rudimentale, è la seguente:

```

1  <html>
2      <head>
3          <title>Benvenuto!</title>
4      </head>
5      <body>
6
7          <br> La mia prima vera pagina in PHP.<br><hr>
8
9          <br><br> Informazioni sul server:
10         <br>Sei giunto su: <?=$SERVER_NAME ?>
11         <br>Stai eseguendo lo script: <?=$SCRIPT_NAME ?>
12
13         <br><br> Esaminiamo il client:
14
15         <br> Indirizzo IP: <?=$REMOTE_ADDR ?>
16         <br> Vedo che provieni da: <?=$HTTP_REFERER ?>
17         <br> Tipo di browser: <?=$HTTP_USER_AGENT ?>
18
19     </body>
20 </html>
```

Salvato il file nella directory del servizio HTTP e richiamato dal navigatore si ottiene qualcosa simile a:

La mia prima vera pagina in PHP.

Informazioni sul server:

¹Nel caso in cui il server sia Apache, le variabili HTTP nella pagina `'phpinfo()'` vengono raccolte nella sezione «variabili di Apache»

²Se si segue il percorso degli script proposti dalla guida, il visitatore arriverà da `<http://www.urcanet.it/brdp/php_manual/esempi/cap_1/primo.php>`

```
Sei giunto su: www.urcanet.it
Stai eseguendo lo script: /brdp/php_manual/esempi/cap_2/benvenuto.php

Esaminiamo il client:
Indirizzo IP: 127.0.0.1
Vedo che provieni da: http://www.urcanet.it/brdp/php_manual/esempi/cap_1/primo.php
Tipo di browser: Mozilla/5.0 (X11; U; Linux 2.4.2-2 i686; en-US; 0.7) Gecko/20010316
```

Anche questo esempio è disponibile in rete all'indirizzo: `<http://www.urcanet.it/brdp/php_manual/esempi/cap_2/benvenuto.php>`

Il risultato potrebbe essere incomprensibile per i visitatori meno esperti. Ad esempio, il tipo di navigatore non è molto chiaro.

Gli strumenti fin qui trattati non permettono di fare molto per migliorare il risultato dello script. Più avanti questo esempio potrà essere ripreso per manipolare il risultato in modo da mostrare al visitatore dei messaggi più chiari.

2.3 Tipi di dati

Il PHP gestisce quattro tipi *scalari*, due tipi *composti* e due tipi *speciali* di dati.

Tabella 2.1. Questi i tipi di dati gestiti dal PHP.

Scalari	Composti	Speciali
'boolean'	'array'	'resource'
'integer'	'object'	'NULL'
'double'		
'string'		

Come già accennato in precedenza, il tipo di una variabile può essere modificato in ogni momento con la funzione `'settype($variabile, tipo)'` che restituisce un valore booleano vero o falso rappresentante l'esito dell'operazione. I possibili tipi da assegnare alla variabile sono:

- `'integer'`
- `'double'`
- `'string'`
- `'array'`
- `'object'`

I criteri di conversione dei tipi sono molto importanti, il codice mostrato di seguito non provoca problemi.

```
0  <?
1  $a = 231;    // da questo momento in poi $a è un intero
2  if(settype($a,double)){
3      /*
4          Se l'operazione è riuscita il valore è TRUE quindi
5          visualizza il testo seguente. Altrimenti salta alla
6          fine dell'if riga 10.
7      */
8      echo "<br>valore settato a double. Ecco il nuovo valore: ";
9      echo $a;
10 }
11 ?>
```

Spiacevoli inconvenienti potrebbero verificarsi se si prova a convertire una stringa in un intero. Per una trattazione più approfondita di questo argomento si rimanda al manuale ufficiale.

Un'altra funzione utile per la gestione dei tipi di dati è `'gettype($variabile)'` che restituisce una stringa contenente il tipo della variabile a essa fornita. Questi i possibili valori restituiti dalla funzione:

- `'integer'`
- `'double'`
- `'string'`
- `'array'`
- `'object'`
- `'unknown type'`

Ecco un semplice esempio di utilizzo della funzione `'gettype()'`:

```
0  <?
1  $a = "brdp";
2  /*
3   la variabile $a è stata
4   inizializzata come stringa
5  */
6  $tipo_a = gettype($a);
7  echo "<br>La variabile è di tipo:";
8  echo $tipo_a;
9  ?>
```

Il risultato dello script sarà:

```
La variabile è di tipo: string
```

Dopo questa breve trattazione sulla gestione dei tipi di dati a disposizione nel PHP, è bene accennare alle caratteristiche di ognuno di essi.

2.3.1 integer

Il tipo di dati `'integer'` rappresenta tutti i numeri dell'insieme matematico degli interi.

$$\mathbb{Z} = \{ \dots, -2, -1, 0, 1, 2, \dots \}$$

Quindi tutti i numeri interi, sia negativi che positivi, vengono gestiti dal PHP come tipi `'integer'`.

Anche i corrispondenti valori in base otto e sedici vengono gestiti come gli interi. Ecco alcuni esempi tratti dal manuale ufficiale:

```
<?
  $a = 1234;      // numero intero positivo
  $a = -123;      // numero intero negativo
  $a = 0123;      // numero ottale (equivalente a 83 in decimale)
  $a = 0x1A;      // numero esadecimale (equivalente a 26 in decimale)
?>
```

Il limite massimo della dimensione dei valori interi dipende dall'architettura dell'elaboratore su cui si lavora.

Nel PHP non esiste un tipo specifico per i numeri naturali.³ Essi vengono gestiti come `'integer'`.

³I numeri naturali sono il sottoinsieme positivo dei numeri interi.

2.3.2 boolean

Il tipo di dati **'boolean'** può assumere solo due valori, vero o falso.

Questo tipo viene utilizzato frequentemente per la gestione degli errori. Molte funzioni, infatti, restituiscono un valore booleano che assume valore vero se l'operazione è andata a buon fine, falso se si è verificato un errore.

Per assegnare il valore vero o falso a una variabile è sufficiente l'istruzione:

```
$bit = True;
```

In questo esempio alla variabile **'\$bit'** è stato assegnato il valore vero. Anche se l'istruzione **'if'** non è ancora stata trattata, è necessario anticipare le metodologie di controllo sul valore della variabile.

In PHP non è necessario (anche se è possibile) eseguire il controllo in questo modo:

```
<?
    $bit = True;

    if($bit == "True"){
        // $bit è vero
        echo " il valore è vero!";
    }
?>
```

Basta scrivere:

```
<?
    $bit = True;

    if($bit){
        // $bit è vero
        echo " il valore è vero!";
    }
?>
```

L'interprete riconosce il tipo di dato e controlla se è vero. Nei prossimi capitoli verranno approfondite le caratteristiche dell'istruzione **'if'**.

2.3.3 double

Come è stato più volte ripetuto, il PHP non necessita della definizione dei tipi di dati che si vanno a utilizzare ma inizializza il tipo di variabile in base al primo valore a essa associato. I «numeri a virgola mobile» in PHP contengono i tipi conosciuti in altri linguaggi come: **'floats'**, **'doubles'**, **'real'**. È possibile inizializzare una variabile di tipo **'double'** assegnandole valori formattati in uno dei seguenti modi:

```
<?
    $a = 1.234;
    $a = 1.2e3;
    $a = 7E-10;
?>
```

Anche per questo tipo, come per gli interi, la dimensione massima dipende dall'architettura dell'elaboratore su cui viene eseguito lo script.

2.3.4 string

Una stringa è un insieme di caratteri. In PHP non ci sono limiti particolari di lunghezza per i dati di tipo **'string'**.

Ci sono più modi di delimitare una stringa. Ecco i due più comuni:

```
0 <?
1 // single quoted string
2 $nome = 'Gianluca Giusti';
3
4 // double quoted string
5 $nome = "Gianluca Giusti";
6
7 /* In entrambi i casi $nome viene inizializzata
8    come stringa dall'interprete PHP */
9 ?>
```

L'interprete PHP riconosce come carattere di escape il **'\'** (barra obliqua inversa).⁴

Di seguito riportiamo i caratteri più comuni, una tabella completa è contenuta nel manuale ufficiale.

Tabella 2.2. Alcuni caratteri di escape per le stringhe PHP.

Carattere	Significato
\t	carattere tabulazione
\n	carattere di fine riga
\\	'\' barra obliqua inversa
\"	'\"' doppio apice
\\$	'\$' dollaro

Ecco alcuni esempi di come gestire le stringhe.

```
0 <?
1 // Valorizzo una variabile che servirà in seguito.
2 $email = " brdp @ urcanet.it ";
3
4 // Stringa semplice:
5 $stringa = "Ecco. Questa è una stringa";
6
7 // Ad essa si possono concatenare altre stringhe alla fine
8 $stringa = $stringa." con dell'altro testo aggiunto";
9
10 // che equivale a scrivere
11 $stringa .= " con dell'altro testo aggiunto";
12
13 // Oppure all'inizio
14 $stringa = "altro testo ancora ".$stringa;
15
16 // Adesso si prende la prima lettera della stringa
17 $prima = $stringa{0};
18
19 // Concatenazione multipla
20 $stringa = $prima." - ".$stringa." <br><br> il mio email: ".$email;
21
22 /* Test sui caratteri di escape. All'interno della stringa posso
23    visualizzare il valore di una variabile ma per visualizzarne
24    il nome devo "inibire" il carattere speciale $. Lo faccio con
25    l'uso della barra obliqua inversa. Ecco come: */
26 $stringa = "Questo il valore di \$email: $email";
27
```

⁴Il carattere di escape viene utilizzato per rappresentare caratteri speciali all'interno di una stringa, caratteri che potrebbero creare confusione all'interprete.

```

28 // c'è differenza con:
29 $stringa = "Questo il valore di $email: $email";
30 ?>

```

L'esempio precedente non è completo. Per poter osservare i risultati delle varie operazioni a schermo è necessario modificare lo script. Questo semplice, ultimo passo è lasciato al lettore come esercizio.

Il PHP fornisce un gran numero di funzioni dedicate alla gestione delle stringhe; uno dei punti di forza di questo linguaggio sta proprio nella semplicità con cui queste possono essere manipolate. Risulta inutile dilungarsi nella trattazione delle singole funzioni, esse verranno descritte negli esempi in cui saranno impiegate. Un elenco completo e dettagliato è contenuto nel manuale ufficiale alla sezione «String functions».

2.3.5 array

Gli array possono essere creati tramite il costrutto `'array()'` oppure tramite la valorizzazione degli elementi.

A differenza dei classici linguaggi di programmazione, il PHP permette di indicizzare gli array non solo mediante interi non negativi, ma anche tramite stringhe. Per chi conosce il Perl il concetto è molto simile a quello degli array associativi.

Di seguito la sintassi per creare un array tramite il costrutto `'array()'`.

```

0  <?
1  array( [chiave =>] valore
2          , ...
3          )
4  // la chiave può essere un intero non negativo o una stringa
5  // il valore può essere qualunque
6  ?>

```

La chiave è contenuta tra le parentesi quadre perchè può essere omessa, se non viene specificata viene incrementato il valore intero. La sintassi di associazione è molto semplice ed intuitiva, per le chiavi intere positive: `'chiave => valore'`, mentre per le chiavi di tipo stringa vanno aggiunte le doppie virgolette `'"chiave" => valore'`.

Per visualizzare il contenuto dell'array è possibile utilizzare la semplice istruzione `'print_r($array)'`. Negli esempi seguenti si vedrà come questa funzione visualizza l'array.

Ecco alcuni esempi.

```

0  <?
1  $a = array( "a", "b", 44, "d", "e");
2  print_r($a);
3  ?>

```

L'istruzione `'print_r($a)'` visualizzerà sullo schermo la struttura e il contenuto dell'array nel seguente modo:

```
Array ( [0] => a [1] => b [2] => 44 [3] => d [4] => e )
```

L'indice dei valori è stato incrementato automaticamente. È bene tenere sempre presente che le chiavi degli array partono da 0 e non da 1.

C'è la possibilità di specificare solo alcune chiavi e lasciare la gestione degli indici omessi all'interprete.

```

0  <?
1  $a = array( "a", "b", "c", "d", 8=>"e", 4=>"f", "g", 3=>"h");
2  print_r($a);
3  ?>

```

Questo il risultato dello script precedente:

```
Array ( [0] => a [1] => b [2] => c [3] => h [8] => e [4] => f [9] => g )
```

A prima vista il risultato può sembrare errato, ma non è così. L'interprete incrementa automaticamente gli indici omessi dei primi quattro valori, ossia da 0 a 3, e mano a mano valorizza l'array. La lettera 'e' va invece inserita nella cella con chiave, specificata, pari a 8 e la 'f' in quella con chiave 4. Per la lettera 'g' non viene specificata nessuna chiave, l'interprete, quindi, riparte automaticamente dall'indice intero più alto incrementandolo di uno e ottiene il valore 9. Rimane da inserire l'ultima lettera 'h' che va, come richiesto, nella cella con chiave pari a 3. Questa operazione sovrascrive la lettera 'd' che già era stata inserita, in automatico, con chiave pari a 3. A questo punto dovrebbero essere chiari i motivi dell'assenza della lettera 'd' e del particolare ordine con cui vengono visualizzati i valori contenuti nell'array.

Un array può essere creato anche senza l'utilizzo del costrutto `'array()'`, in questo caso la sintassi ricorda quella dei più comuni linguaggi di programmazione.

Di seguito sono riportati gli esempi precedenti con l'utilizzo del secondo metodo di creazione.

```
0  <?
1  $a[] = "a";
2  $a[] = "b";
3  $a[] = 44;
4  $a[] = "d";
5  $a[] = "e";
6  print_r($a);
7  ?>
```

Il secondo esempio può essere tradotto in questo modo:

```
0  <?
1  $a[] = "a";
2  $a[] = "b";
3  $a[] = "c";
4  $a[] = "d";
5  $a[8] = "e";
6  $a[4] = "f";
7  $a[] = "g";
8  $a[3] = "h";
9  print_r($a);
10 ?>
```

I due procedimenti per la creazione degli array sono equivalenti.

Gli array possono avere chiavi miste, come nell'esempio seguente, in cui alcune sono intere e alcune stringhe.

```
0  <?
1  $a["rosso"] = "a";
2  $a[] = "c";
3  $a[8] = "e";
4  $a["nero"] = "f";
5  $a[] = "g";
6  print_r($a);
7  ?>
```

Ecco il risultato:

```
Array ( [rosso] => a [0] => c [8] => e [nero] => f [9] => g )
```

È interessante studiare una possibile applicazione pratica degli array. Nel seguente esempio verrà ripresa la funzione `'date()'` già incontrata nella sezione 1.1.

```
0  <?
1  // valorizzo l'array dei giorni della settimana con il metodo classico.
```

```

2  $giorno[0] = "Domenica";
3  $giorno[1] = "Lunedì";
4  $giorno[2] = "Martedì";
5  $giorno[3] = "Mercoledì";
6  $giorno[4] = "Giovedì";
7  $giorno[5] = "Venerdì";
8  $giorno[6] = "Sabato";
9
10 // valorizzo l'array dei mesi dell'anno con il costrutto array()
11 $mese = array(
12     1 => "Gennaio",
13     2 => "Febbraio",
14     3 => "Marzo",
15     4 => "Aprile",
16     5 => "Maggio",
17     6 => "Giugno",
18     7 => "Luglio",
19     8 => "Agosto",
20     9 => "Settembre",
21     10 => "Ottobre",
22     11 => "Novembre",
23     12 => "Dicembre"
24 );
25 // Prendo il mese in formato numerico da 1 a 12.
26 $numero_mese = date("n");
27
28 /* Prendo il giorno della settimana da 0 (domenica) a 6 (sabato)
29    questa volta formato tutto annidando più funzioni.
30    in PHP è possibile! */
31 $giorno_settimana = $giorno[date("w")];
32
33 // Formatto la data nel modo: Lunedì 19 Novembre 2001
34 $oggi = $giorno_settimana." ".date("d")."-".$mese[$numero_mese]."-".date("Y");
35
36 // Visualizzo la data a schermo concatenandola ad una stringa
37 echo "<br> Oggi è: <b>".$oggi."</b>";
38 ?>

```

Il risultato di questo script, raggiungibile presso http://www.urcanet.it/brdp/php_manual/esempi/cap_2/data.php, è:

```
Oggi è: Domenica 18-Novembre-2001
```

Gli array rappresentano una delle strutture più versatili in PHP, a differenza dei linguaggi classici, infatti, la dimensione non deve essere specificata a priori. Questo permette una dinamicità e una libertà di utilizzo notevole.

Il PHP gestisce anche gli array multidimensionali e gli array annidati.

Non si entrerà nel merito, si darà per scontata la teoria sulla gestione degli array, che è simile per i diversi tipi di linguaggi, mentre si tratterà la sintassi tramite uno script di esempio.

Il seguente è un esempio sull'utilizzo degli array multidimensionali.

```

0  <html>
1  <head>
2      <title>Semplice Agenda telefonica statica</title>
3  </head>
4  <body>
5
6      <?
7          /*
8              Un semplice esempio di array multidimensionale.
9              Una rubrica telefonica.
10             */
11
12         $a["nome"][0] = "Gianluca";

```

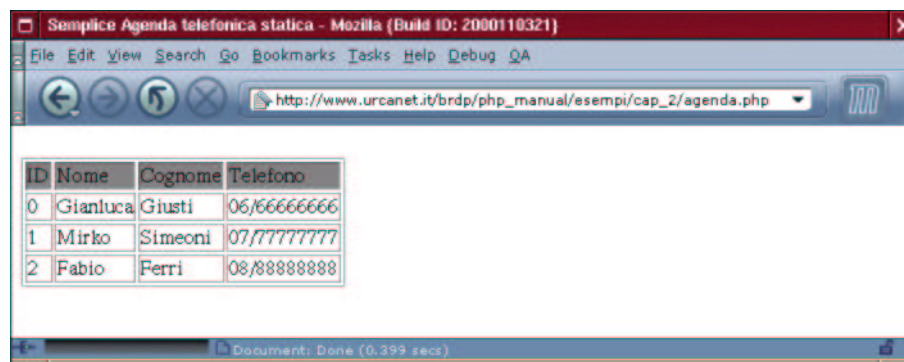
```

13     $a["cognome"][0] = "Giusti";
14     $a["tel"][0] = "06/66666666";
15
16     $a["nome"][1] = "Mirko";
17     $a["cognome"][1] = "Simeoni";
18     $a["tel"][1] = "07/77777777";
19
20     $a["nome"][2] = "Fabio";
21     $a["cognome"][2] = "Ferri";
22     $a["tel"][2] = "08/88888888";
23
24     /*
25         Adesso elenchiamo la rubrica. Lo faremo senza
26         utilizzare Nessuna struttura ciclica per non
27         confondere le idee
28     */
29
30     ?>
31
32     <br>
33
34     <table border="1">
35         <tr bgcolor="gray" >
36             <td>ID</td>
37             <td>Nome</td>
38             <td>Cognome</td>
39             <td>Telefono</td>
40         </tr>
41         <tr>
42             <td>0</td>
43             <td><?=$a[nome][0]?></td>
44             <td><?=$a[cognome][0]?></td>
45             <td><?=$a[tel][0]?></td>
46         </tr>
47         <tr>
48             <td>1</td>
49             <td><?=$a[nome][1]?></td>
50             <td><?=$a[cognome][1]?></td>
51             <td><?=$a[tel][1]?></td>
52         </tr>
53         <tr>
54             <td>2</td>
55             <td><?=$a[nome][2]?></td>
56             <td><?=$a[cognome][2]?></td>
57             <td><?=$a[tel][2]?></td>
58         </tr>
59     </table>
60
61     </body>
62 </html>

```

Una volta salvato il codice nella propria directory dei documenti del servizio HTTP si può verificare il risultato. Anche questo esempio è disponibile all'indirizzo: http://www.urcanet.it/brdp/php_manual/esempi/cap_2/agenda.php, inoltre, il risultato è riportato in figura 2.1.

Figura 2.1. La rubrica telefonica è stata formattata in una tabella HTML.



Inutile dire che questo è solo un'esempio per prendere confidenza con la sintassi PHP relativa agli array e che esistono soluzioni migliori per il salvataggio dei dati. Più avanti nella guida si tratteranno i file e le basi di dati.

Un esercizio molto utile lasciato al lettore è la realizzazione di uno script clone del precedente con l'utilizzo del costrutto `'array()'` per la gestione dell'array utilizzato nella rubrica telefonica.

Lo studio degli array di array viene momentaneamente lasciato al lettore come approfondimento, con l'impegno di tornare sull'argomento appena conclusi i punti fondamentali di questa guida.

2.3.6 object

Una classe è una collezione di variabili e di funzioni dette metodi. Una volta definita la classe è possibile istanziare uno o più oggetti della stessa classe. Ognuno di essi è indipendente dagli altri.

Una trattazione approfondita può essere trovata nella sezione 6 di questa guida.

2.4 Operatori

Il PHP gestisce numerosi tipi di operatori, di seguito sono elencate le caratteristiche principali dei tipi utilizzati più di frequente. Per una trattazione completa si rimanda il lettore al manuale ufficiale.

Tabella 2.3. Questi alcuni degli operatori gestiti dal PHP.

OPERATORI	
aritmetici	assegnazione
controllo degli errori	logici
incremento e decremento	confronto

Verranno analizzati singolarmente gli operatori riportati nella tabella 2.3.

2.4.1 Operatori aritmetici

Nella tabella 2.4 sono riportati gli operatori aritmetici con le rispettive caratteristiche.

Tabella 2.4. Le caratteristiche degli operatori aritmetici.

Esempio	Nome	Risultato
$\$a + \b	Addizione	Somma tra $\$a$ e $\$b$
$\$a - \b	Sottrazione	Differenza tra $\$a$ e $\$b$
$\$a * \b	Moltiplicazione	Prodotto tra $\$a$ e $\$b$
$\$a / \b	Divisione	Quoziente tra $\$a$ e $\$b$

Lo schema è stato preso dal manuale ufficiale, è molto chiaro e non ha bisogno di molte spiegazioni. Bisogna tenere ben presente che il tipo della variabile a cui si assegna il risultato dipende dal tipo dei due valori ' $\$a$ ' e ' $\$b$ '.

2.4.2 Operatori di assegnazione

L'operatore fondamentale per l'assegnazione di un valore ad una variabile è '='. La sintassi fondamentale è:

```
0  <?
1  // assegno ad $a il valore intero 123
2  $a = 123;
3
4  // assegno a $b il contenuto di $a
5  $b = $a;
6
7  // assegno una stringa alla variabile $c
8  $c = "questa è una stringa";
9  ?>
```

L'operatore '=' può essere utilizzato insieme ad altri operatori elencati nell'esempio:

```
0  <?
1  // assegno ad $a un valore intero
2  $a = 2;
3
4  // assegno a $b un altro valore intero
5  $b = 5;
6
7  // posso assegnare a $a la somma di $a e $b in questo modo
8  $a = $a + $b;
9
10 // oppure in questo modo
11 $a += $b;
12
13 // stesso discorso per tutti gli operatori aritmetici
14 $a -= $b;
15 $a *= $b;
16 $a /= $b;
17
18 // posso anche assegnare un valore a più variabili
19 $a = $b = 30;
20
21 // o ancora in questo esempio sia $a che $b valgono 6
22 $a = 3;
23 $b = $a += 3;
24
25 // per le stringhe. $a varrà "ciao a tutti!!!"
26 $a = "ciao a";
27 $a .= " tutti!!!";
28
29 // equivale a concatenare la stringa in questo modo
30 $a = "ciao a";
31 $a = $a." tutti!!!";
32 ?>
```


I commenti inseriti nell'esempio dovrebbero bastare a descrivere la versatilità degli operatori appena trattati.

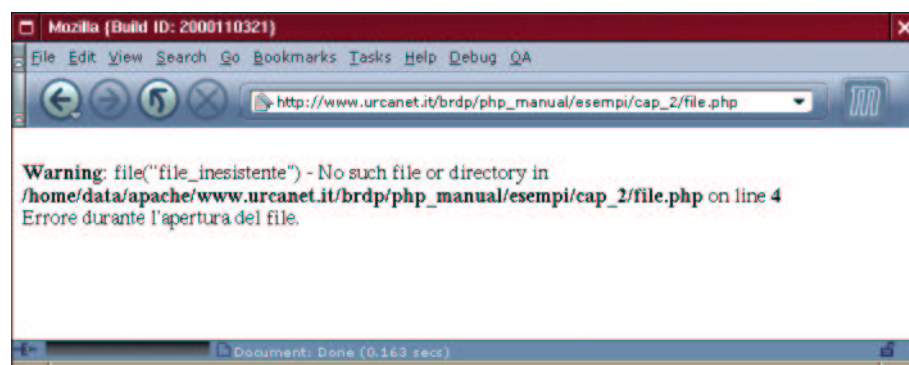
2.4.3 Operatori di controllo degli errori

Nell'esempio seguente viene gestito un errore dovuto al tentativo di lettura di un file inesistente. Con l'occasione viene utilizzata la funzione PHP `'file('nome_del_file')'` che restituisce un array contenente le righe del file passato alla funzione.

```
0  <?
1  // provo ad aprire un file che non esiste
2  $file_array = file('file_inesistente') or
3      die (" Errore durante l'apertura del file. ");
4
5  /*
6      il file non esiste, lo script si ferma
7      mostrando i messaggi di errore
8  */
9  ?>
```

Se si salva il codice in un file e si richiama tramite il navigatore, si nota che oltre al messaggio di errore inserito nello script viene visualizzato un messaggio introdotto dall'interprete PHP. I messaggi di errore possono contenere informazioni utili ai male intenzionati, è bene, quindi, gestire tali messaggi al meglio. Un esempio è riportato in figura 2.2 dove si nota il percorso su file system dello script di prova.

Figura 2.2. Ecco come appare la pagina contenente l'errore non gestito. Oltre al messaggio di errore da noi inserito viene visualizzato anche un warning dall'interprete PHP.



Un modo molto semplice per non far visualizzare i messaggi d'errore dell'interprete è premettere il carattere '@' alla funzione. Ad esempio: `@file('dati.txt')`.

L'esempio precedente potrebbe essere modificato in questo modo:

```
0  <?
1  // provo ad aprire un file che non esiste
2  $file_array = @file('file_inesistente') or
3      die (" Errore durante l'apertura del file. ");
4
5  /*
6      In questo caso il messaggio di errore dell'interprete
7      PHP è stato silenziato dalla '@' premessa alla funzione file().
8      Verrà quindi mostrato solo il nostro messaggio di errore
9  */
10 ?>
```

Il costrutto `'die()'` non fa altro che terminare l'esecuzione del programma, in caso di errore, mostrando il messaggio contenuto tra le parentesi.

2.4.4 Operatori di incremento e decremento

Come detto, la sintassi PHP ricorda molto quella del C. Gli operatori di pre e post incrementazione sono gestiti come nel linguaggio C.

Nella tabella 2.5 sono riportati i quattro operatori:

Tabella 2.5. Questi gli operatori di incremento e decremento gestiti dal PHP.

Esempio	Nome	Risultato
'\$a++'	Post-incremento	Restituisce '\$a' e poi la incrementa di uno
'++\$a'	Pre-incremento	Incrementa di uno '\$a' e poi la restituisce
'\$a--'	Post-decremento	Restituisce '\$a' e poi la decrementa di uno
'--\$a'	Pre-decremento	Decrementa di uno '\$a' e poi la restituisce

Nell'esempio seguente vengono utilizzati i quattro operatori:

```
0  <?
1    $a = 1;
2
3    // Operatori di pre-incremento
4
5    echo "<br>La variabile vale: ".$a;
6    echo " e dopo vale: ".$a;
7
8    // Operatori di post-incremento
9
10   echo "<br>La variabile vale: ".$a++;
11   echo " e dopo vale: ".$a;
12
13   // Operatori di pre-decremento
14
15   echo "<br>La variabile vale: ".$a--;
16   echo " e dopo vale: ".$a;
17
18   // Operatori di post-decremento
19
20   echo "<br> La variabile vale: ".$a--;
21   echo " e dopo vale: ".$a;
22
23  ?>
```

Se si prova a salvare l'esempio in un file nella directory del servizio HTTP, una volta richiamato mediante il navigatore, il risultato sarà di questo tipo:

```
La variabile vale: 2 e dopo vale: 2
La variabile vale: 2 e dopo vale: 3
La variabile vale: 2 e dopo vale: 2
La variabile vale: 2 e dopo vale: 1
```

Ragionando passo passo si capisce il lavoro dell'interprete e di conseguenza il risultato ottenuto. La variabile inizialmente vale 1. Nella riga 5 viene incrementata, prima di essere visualizzata, tramite l'operatore '++\$a'. Nella riga 6 viene mostrato il valore di '\$a' senza eseguire alcuna operazione. Da queste prime due righe di codice si ottiene la riga:

```
La variabile vale: 2 e dopo vale: 2
```

Nella riga 10, invece, viene utilizzato l'operatore '\$a++' che prima visualizza il contenuto della variabile e poi lo incrementa. Infatti la riga 11 restituisce un valore di '\$a' pari a 3. Il risultato è evidente nella seconda riga del file HTML generato dallo script di prova.

Stessa procedura per gli operatori di pre e post decremento, fino a ritornare al valore iniziale della variabile.

2.4.5 Operatori logici

Gli operatori logici gestiti dal PHP sono riportati nella tabella 2.6.

Tabella 2.6. Questi gli operatori logici gestiti dal PHP.

Esempio	Nome	Risultato
'\$a and \$b'	AND	vera se \$a e \$b sono vere
'\$a or \$b'	OR	vera se \$a o \$b è vera
'\$a xor \$b'	XOR	vera se \$a o \$b è vera ma non entrambe
'!\$a'	NOT	Negazione. Vera se \$a non è vera
'\$a && \$b'	AND	Simile a 'and' ma con precedenza diversa
'\$a \$b'	OR	Simile a 'or' ma con precedenza diversa

Le precedenze degli operatori sono riportate nel manuale ufficiale.

In questo momento è inutile dilungarsi in esempi, gli operatori logici verranno approfonditi in seguito. In particolare verranno trattati con l'introduzione dell'istruzione 'if'.

2.4.6 Operatori di confronto

Il PHP gestisce tutti gli operatori di confronto riportati nella tabella 2.7.

Tabella 2.7. Gli operatori di confronto gestiti dal PHP.

Esempio	Nome	Risultato
'\$a == \$b'	Uguale	vera se \$a è uguale a \$b
'\$a === \$b'	Identico	vera se \$a è uguale a \$b e sono dello stesso tipo
'\$a != \$b'	Diverso	vera se \$a è diverso da \$b
'\$a <> \$b'	Diverso	vera se \$a è diverso da \$b
'\$a !== \$b'	Non Identico	vera se \$a non è uguale a \$b o non sono dello stesso tipo
'\$a < \$b'	Minore	vera se \$a è minore di \$b
'\$a > \$b'	Maggiore	vera se \$a è maggiore di \$b
'\$a <= \$b'	Minore o uguale	vera se \$a è minore o uguale a \$b
'\$a >= \$b'	Maggiore o uguale	vera se \$a è maggiore o uguale a \$b

Esempi pratici sugli operatori di confronto verranno trattati in seguito, nella guida, con l'introduzione delle «strutture di controllo».

2.5 Strutture di controllo

Nella tabella 2.8 sono riportate le strutture di controllo che verranno trattate in dettaglio nelle pagine seguenti.

Tabella 2.8. Le strutture di controllo.

Strutture	di controllo	
if	while	break
else	do..while	include()
elseif	for	
switch	foreach	

le strutture di controllo sono state raggruppate per tipologia e ordinate secondo la difficoltà. Di seguito verranno trattati prima i costrutti più semplici e poi quelli più complessi.

Nella prima colonna sono raccolte le strutture tramite le quali si è in grado di eseguire determinate

istruzioni al verificarsi di particolari condizioni.

Nella seconda colonna sono raccolte le strutture mediante cui è possibile realizzare e gestire delle operazioni cicliche, fino a quando una particolare condizione viene soddisfatta e interrompe il ciclo.

In questo ultimo caso è importante tenere sempre presente la condizione di STOP. Questo per evitare dei cicli infiniti che possono bloccare il funzionamento del programma.

Inoltre negli esempi che seguono verranno utilizzati gli operatori precedentemente introdotti.

2.5.1 if

Il costrutto **'if'** è molto importante e utilizzato in tutti i linguaggi di programmazione. La sintassi è la seguente:

```
if(condizione)
    singola istruzione
```

Utilizzato in questo modo, solamente un'istruzione è condizionata dall'esito della condizione contenuta nel costrutto **'if'**. Ecco un esempio:

```
0 <?
1  $a = 3;
2  $b = 5;
3
4  if($a < $b) // uso un operatore di confronto
5      echo "Condizionato da if. Solo se \"$a\" è minore di \"$b.\"";
6
7      echo "<br>Questo, invece, viene scritto comunque!!!";
8
9 ?>
```

Se si ha bisogno di condizionare una o più istruzioni la sintassi da utilizzare è la seguente:

```
if(condizione){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}
```

Riprendendo l'esempio precedente si può scrivere:

```
0 <?
1  $a = 3;
2  $b = 5;
3
4  if($a < $b){
5      echo "Istruzione 1.";
6      echo "Istruzione 2.";
7      echo "Istruzione 3.";
8  }
9
10 echo "<br>Questo, invece, viene scritto comunque!!!";
11
12 ?>
```

La differenza è semplice e lampante. Il blocco delle istruzioni condizionate va contenuto tra le parentesi **'{ }'**.

2.5.2 else

Nell'ultimo esempio viene gestito un solo evento. Se si volesse gestire anche l'evento `$a >= $b` si potrebbe aggiungere, in modo poco elegante e funzionale, un altro `'if($a >= $b){ istruzioni }'`. Tuttavia la soluzione più corretta è sicuramente l'utilizzo della struttura `'else'`

La sintassi del costrutto `'else'` è la seguente:

```
if(condizione){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}
else{
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione m
}
```

Di seguito sono riportati alcuni esempi per chiarirne l'utilizzo e per prendere confidenza con gli operatori precedentemente trattati.

```
0  <?
1
2      if(!$a){ // $a è falso.
3          $a = "ok";
4      }else{
5          $a = "ko";
6      }
7
8      // a questo punto $a vale "ok"
9
10     if($a == "ok"){
11         echo "Sono entrato nell'if e \ $a vale: ".$a;
12         echo "<br> Adesso cambiamo il contenuto di \ $a";
13         $a = 17;
14     }else{
15         echo "\ $a non vale \"ok\" e quindi mi trovo nell'else. \ $a: ".$a;
16         echo "<br> anche in questo caso setto \ $a uguale a 17 ma in modo diverso.";
17         $a = 16;
18         $a += 1;
19     }
20
21     // adesso scrivo il tipo e il valore di $a
22
23     if(is_integer($a)){
24         echo "<br> \ $a è un INTERO e vale: ".$a;
25     }else{
26         echo "<br> \ $a NON è un intero e vale: ".$a;
27     }
28
29  ?>
```

Lo script, così come nell'esempio, restituisce un risultato simile al seguente:

```
Sono entrato nell'if e $a vale: ok
Adesso cambiamo il contenuto di $a
```

`$a` è un `INTERO` e vale: 17

Per comprendere al meglio il flusso del programma e delle strutture in esso contenute, si può inserire nella riga 1 la valorizzazione di `'$a'`, ad esempio:

```
1      $a = "qualche valore";
```

Così facendo l'esito del primo `'if'` cambia e di conseguenza tutto il resto dello script.

Con l'occasione si è utilizzata una nuova funzione, `'is_integer()'`, che restituisce un valore booleano vero se la variabile ad essa fornita è di tipo intero, falso se la variabile non è intera.

Esistono altre funzioni simili per verificare il tipo delle variabili, tra cui: `'is_array()'`, `'is_double()'`, `'is_string()'`. Il funzionamento è analogo a quello di `'is_integer()'`. Ovviamente esistono altre funzioni di questo tipo e sono raccolte nella sezione «Funzioni di Variabili» del manuale ufficiale del PHP.

2.5.3 elseif

Si supponga di dover gestire il confronto tra due variabili `'$a'` e `'$b'` di tipo intero. I possibili risultati del confronto sono tre:

- `$a > $b`
- `$a < $b`
- `$a = $b`

Utilizzando le strutture viste fino ad ora, non si è in grado di risolvere il problema. Una soluzione poco elegante, e sicuramente non ottimale, è eseguire tre `'if'` in cascata.

Per risolvere problemi di questo tipo il PHP mette a disposizione la struttura `'elseif'`. La sintassi è analoga a quella di `'else'`.

```
if(condizione){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}elseif(condizione){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione m
}else{
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione k
}
```

Ecco come applicare la sintassi all'esempio precedentemente descritto:

```

0  <?
1  $a = 3; // valorizzo la variabile $a a mio piacere
2  $b = 6; // valorizzo la variabile $b a mio piacere
3
4  // adesso eseguo il confronto con un unica istruzione elseif
5
6  if ($a > $b) {
7      echo " \ $a è maggiore di \ $b ";
8  } elseif ($a == $b) {
9      echo " \ $a è uguale a \ $b ";
10 } else {
11     echo " \ $a è minore di \ $b ";
12 }
13
14 ?>

```

Dunque in questo modo si è in grado di gestire più condizioni. Il flusso è molto semplice e intuitivo. Se una delle due condizioni è soddisfatta l'interprete esegue le istruzioni ad essa associate ed esce dal blocco **'elseif'**. Se è presente un **'else'** finale e nessuna condizione è stata soddisfatta, l'interprete ne esegue le istruzioni e lascia il blocco **'elseif'**.

Va precisato che viene eseguito al massimo un blocco di istruzioni. Se più condizioni sono soddisfatte, solo le istruzioni associate alla prima condizione vera vengono eseguite, le altre vengono saltate.

Si può inserire più di un **'elseif(condizione)'** all'interno della struttura. Vediamo come, modificando l'esempio precedente.

```

0  <?
1  $a = 3; // valorizzo la variabile $a a mio piacere
2  $b = 6; // valorizzo la variabile $b a mio piacere
3
4  // adesso eseguo il confronto con un unica istruzione elseif
5
6  if ($a > $b) {
7      echo " \ $a è maggiore di \ $b ";
8  } elseif ($a === $b) {
9      echo " \ $a è identica a \ $b ";
10 } elseif ($a == $b) {
11     echo " \ $a è uguale a \ $b ";
12 } else {
13     echo " \ $a è minore di \ $b ";
14 }
15
16 ?>

```

Per verificare la correttezza e capire il diverso funzionamento, si provi ad assegnare alla variabile **'\$b'** i seguenti valori:

- \$b = 6;
- \$b = 3;
- \$b = 3.0;

Infine si modifichino le righe interessate allo script di esempio nel seguente modo:

```

8  } elseif ($a == $b) {
9      echo " \ $a è uguale a \ $b ";
10 } elseif ($a === $b) {
11     echo " \ $a è identica a \ $b ";

```

In questo modo la voce **'\$a è identica a \$b'** non viene mai visualizzata. Al lettore il semplice compito di scoprirne il motivo.⁵

2.5.4 switch

Spesso ci si trova a dover confrontare il contenuto di una variabile con più valori, in questi casi è preferibile utilizzare la struttura **'switch'** al posto di **'elseif'**. Un esempio renderà più chiaro il concetto.

Si supponga di dover gestire un menu. In base al valore di una variabile vengono eseguite una o più operazioni. Nell'esempio la variabile **'\$scelta'** è valorizzata nella riga 2. Per provare il funzionamento del programma basterà cambiare il valore a questa variabile.

```
0  <?
1  // valorizzo la variabile $scelta
2  $scelta = 0;
3
4  // con switch gestisco i confronti e le operazioni.
5
6  switch ($scelta) {
7      case 0:
8          echo "<br> lancio l'istruzione associata al valore 0 ";
9          break; // esce da switch
10     case 1:
11         echo "<br> lancio l'istruzione associata al valore 1 ";
12         break; // esce da switch
13     case 2:
14         echo "<br> lancio l'istruzione associata al valore 2 ";
15         break; // esce da switch
16     default:
17         echo "<br> NESSUNA OPERAZIONE ASSOCIATA alla scelta";
18 }
19
20
21 ?>
```

Nella struttura **'switch'** a differenza di **'elseif'**, l'interprete esegue le istruzioni successive al **'case'** soddisfatto. Questo è il motivo per cui si utilizza il comando **'break'**. Più avanti si tratterà questo costrutto, per il momento basta sapere che serve ad abbandonare la struttura di controllo.

L'esempio precedente genera un risultato di questo tipo:

```
lancio l'istruzione associata al valore 0
```

Se non ci fossero i **'break'** in ogni blocco di istruzione il risultato sarebbe:

```
lancio l'istruzione associata al valore 0
lancio l'istruzione associata al valore 1
lancio l'istruzione associata al valore 2
NESSUNA OPERAZIONE ASSOCIATA alla scelta
```

Se il funzionamento di **'switch'** non risulta ancora chiaro, si provi, dopo aver tolto le istruzioni **'break'**, ad assegnare alla variabile **'\$scelta'** i valori: 0,1 e 2, verificandone i singoli risultati.

Nel caso di più uguaglianze la differenza tra **'switch'** e **'elseif'** è minima, in questi casi, l'utilizzo di **'switch'** risulta più elegante e leggibile di **'elseif'**. Di seguito viene riportato l'esempio precedente riscritto utilizzando la struttura **elseif**.

```
0  <?
1  // valorizzo la variabile $scelta
2  $scelta = 0;
```

⁵La condizione di identità include quella di uguaglianza, quindi, se due variabili sono identiche di conseguenza sono anche uguali, mentre non è vero il contrario. Ecco perchè bisogna fare attenzione all'ordine in cui si scrivono le condizioni.


```

3
4    // adesso utilizzo elseif per gestire le operazioni
5
6    if($scelta == 0) {
7        echo "<br> lancio l'istruzione associata al valore 0 ";
8    }elseif($scelta == 1){
9        echo "<br> lancio l'istruzione associata al valore 1 ";
10   }elseif($scelta == 2){
11       echo "<br> lancio l'istruzione associata al valore 2 ";
12   }else{
13       echo "<br> NESSUNA OPERAZIONE ASSOCIATA alla scelta";
14   }
15
16 ?>

```

2.5.5 while

Il **'while'** è la struttura di gestione dei cicli più semplice del PHP. La traduzione in italiano è: "mentre".

La sintassi è la seguente:

```

while(condizione){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}

```

Cioè: «**mentre** la condizione è vera esegui le istruzioni».

Il seguente esempio può aiutare a chiarire le idee.

```

0  <?
1    $i=1;
2    while ($i <= 6){    // mentre $i è minore o uguale a 7
3        echo "<br> \ $i adesso vale: ".$i;
4        $i++;
5    }
6    ?>

```

Dunque, in questo caso la condizione di stop, che fa fermare il ciclo, è '**\$i > 6**'. Ovvero '**\$i <= 6**' è falsa.

Il risultato è il seguente:

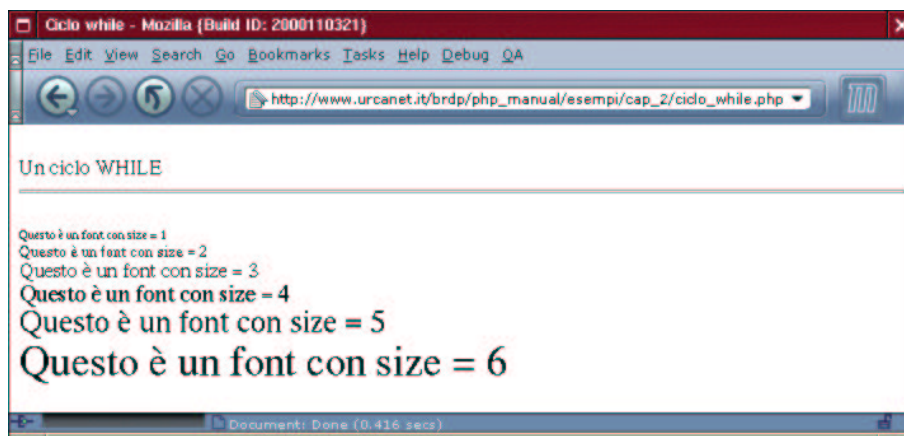
```

$i adesso vale: 1
$i adesso vale: 2
$i adesso vale: 3
$i adesso vale: 4
$i adesso vale: 5
$i adesso vale: 6

```

Tramite un semplice ciclo PHP si può generare una pagina HTML come quella riportata in figura 2.3

Figura 2.3. Questo è stato generato dal ciclo 'while'.



Semplicemente è stato eseguito il seguente codice:

```

0 <html>
1 <head>
2 <title>Ciclo while</title>
3 </head>
4 <body>
5
6 <br> Un ciclo WHILE<br><hr>
7
8 <?
9     $i=1;
10    while ($i <= 6){
11    ?>
12
13        <br> <font size="<?=$i?>">Questo è un font con size = <?=$i?></font>
14
15    <?
16        $i++;
17    }
18    ?>
19
20
21 </body>
22 </html>

```

Dal menu visualizza del navigatore, l'opzione sorgente pagina, mostra il seguente codice:

```

0 <html>
1 <head>
2 <title>Ciclo while</title>
3 </head>
4 <body>
5 <br> Un ciclo WHILE<br><hr>
6
7 <br> <font size="1">Questo è un font con size = 1</font>
8 <br> <font size="2">Questo è un font con size = 2</font>
9 <br> <font size="3">Questo è un font con size = 3</font>
10 <br> <font size="4">Questo è un font con size = 4</font>
11 <br> <font size="5">Questo è un font con size = 5</font>
12 <br> <font size="6">Questo è un font con size = 6</font>
13
14 </body>
15 </html>

```

Le righe dalla 7 alla 12 sono state generate dall'interprete con l'esecuzione del ciclo 'while' dell'esempio.

2.5.6 do..while

A differenza del **'while'** il **'do..while'** esegue il controllo sulla condizione dopo l'esecuzione delle istruzioni. Ecco la sintassi:

```
do {
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}while(condizione);
```

Dunque l'interprete esegue le istruzioni e solo dopo controlla la condizione di stop, se è vera riesegue le istruzioni, se è falsa abbandona il ciclo.

Di seguito è riportato un esempio costituito da due cicli, uno utilizza il costrutto **'do..while'**, l'altro il **'while'**. Le condizioni iniziali e di uscita sono le stesse ma il risultato è diverso. Al lettore il compito di capire tali differenze.

```
0  <?
1    // Condizione iniziale!
2    $inizio = 0;
3
4    // ciclo do..while
5
6    do{
7        echo "<br> ciclo: do..while. \ $inizio vale: ".$inizio;
8    }while($inizio > 0);
9
10   // mentre il ciclo while
11
12   while($inizio > 0){
13       echo "<br> ciclo: while. \ $inizio vale: ".$inizio;
14   }
15  ?>
```

2.5.7 for

La sintassi del ciclo **'for'** può sembrare incomprensibile ad una prima lettura, invece è molto coincisa, versatile e, con un po' di pratica, semplice. Eccola:

```
for(condizione iniziale; condizione stop; variazione parametro){
    istruzione 1
    istruzione 2
    .
    .
    .
    istruzione n
}
```

Tale sintassi viene applicata nel seguente modo:

```
for($i=0; $i<10; $i++){
    echo "<br>tutte le istruzioni che desidero";
}
```

Il codice dell'esempio può essere letto come: «partendo da $i = 0$, mentre $i < 10$ esegui le istruzioni e incrementa i di uno».

Sia le condizioni che la variazione possono essere omesse o incluse tra le istruzioni. Per il momento verrà trattata la sintassi classica. Il modo più semplice e diretto è, come al solito, un esempio ben commentato.

```

0  <?
1      // sintassi classica:
2
3      for($i=0; $i<10; $i++){
4          echo "<br> non serve incrementare, lo fa il for";
5          echo "<br> infatti ecco il valore di \$i: ".$i;
6      }
7
8      // se ho una sola istruzione, come con l'if
9
10     for ($i=0; $i<10; $i++)
11         echo "<br> per una sola va bene così. \$i vale: ".$i;
12
13     // oppure se è solo un print
14
15     for($i=0; $i<10; print"<br>".$i, $i++)
16
17     // Ancora posso incrementare $i tra le istruzioni
18
19     for($i=0; $i<10; ){
20         echo "<br>incrementare \$i altrimenti il ciclo è infinito. \$i=".$i;
21         $i++;
22     }
23
24     // Oppure premetto la condizione iniziale.
25
26     $i=0;
27     for( ; $i<10; $i++){
28         echo "<br>eccolo : ".$i;
29     }
30
31 ?>

```

Come detto è possibile omettere anche le condizioni iniziale e di stop. Per fare questo è necessario utilizzare la struttura **'break'** che verrà introdotta nelle pagine seguenti. In quell'occasione verranno trattate tali eccezioni.

A questo punto, un ottimo esercizio per il lettore è la traduzione degli esempi visti per le strutture del **'while'** e **'do..while'**, in script uguali o simili utilizzando le varie derivazioni del costrutto **'for'**.

2.5.8 foreach

Va subito notato che il costrutto **'foreach'** è stato implementato dalla versione 4.0 in poi e quindi non è gestita dalle versioni precedenti del PHP.

L'uso del **'foreach'** è indicato per la gestione degli array e mette a disposizione due sintassi principali:

```

foreach($variabile_array as $value){
    ....
    istruzioni
    ....
}

```

e:

```

foreach($variabile_array as $key => $value){
    ...
    istruzioni
    ...
}

```

il suo funzionamento, a differenza del semplice **'for'**, non è molto intuitivo ed è vincolato all'utilizzo di un array. Si nota immediatamente che la sintassi non richiede l'utilizzo di indici né di incrementi, infatti il costrutto opererà su tutti gli argomenti dell'array indistintamente. Dunque questo strumento risulta comodo e versatile quando si ha a che fare con array con chiavi particolari o non consecutive (nel caso di chiavi numeriche).

Si supponga di avere un primo array composto da chiavi intere e consecutive simile al seguente:

```
<?
$array_1 = array (1, "a", 5, "c", "ciao"); // con chiavi da 0 a 4
print_r($array_1);
?>
```

e un secondo composto da chiavi miste e quindi non consecutive come questo:

```
<?
$array_2 = array (
    "uno" => 1,           // chiave stringa e valore intero
    "frutta" => "mela",   // chiave stringa e valore stringa
    5 => "cinque",        // chiave intero e valore stringa
    40 => 30              // chiave intero e valore intero
);
print_r($array_2);
?>
```

I risultati dei due script sono semplicemente:

```
Array ( [0] => 1 [1] => a [2] => 5 [3] => c [4] => ciao )
Array ( [uno] => 1 [frutta] => mela [5] => cinque [40] => 30 )
```

Si supponga ora di non conoscere le chiavi del secondo array perchè assegnate dinamicamente da un codice precedente. In questo caso non si sarebbe in grado di operare sull'array con le strutture cicliche classiche perchè non c'è nessuna relazione tra le chiavi. Mentre nel primo la relazione tra le chiavi è semplice, sono interi successivi a partire da 0. In queste situazioni la struttura **'foreach'** è tanto indispensabile quanto semplice ed immediata da utilizzare.

L'esempio potrebbe continuare con la necessità di separare i valori interi da quelli non interi, contenuti in **'\$array_2'**. Di seguito viene risolto il problema con l'utilizzo del costrutto **'foreach'** e della prima sintassi.

```
0  <?
1      foreach ($array_2 as $val) {
2          echo "<br>\$val = ".$val;
3          if(is_integer($val)){
4              echo " -----> ed è un intero...";
5              // possono seguire istruzioni per trattare i dati interi
6          }else{
7              echo " -----> ma non è un intero...";
8              // possono seguire istruzioni per trattare i dati NON interi
9          }
10     }
11 ?>
```

'foreach' in italiano può essere tradotto come **'per ogni'**, dunque la sintassi è ora più chiara: per ogni valore dell'array **'\$array_2'** preso come **'\$val'** esegui le istruzioni associate.

Il costrutto ad ogni ciclo valorizza la variabile **'\$val'** con un elemento dell'array. Ripete l'operazione per tutti gli elementi indipendentemente dalle chiavi ad essi associati.

Nell'esempio le cose sono state complicate con l'annidamento di un **'if'** che verifica ad ogni ciclo il tipo di dato estratto dall'array, il risultato è il seguente:

```
$val = 1 -----> ed è un intero...
$val = mela -----> ma non è un intero...
$val = cinque -----> ma non è un intero...
$val = 30 -----> ed è un intero...
```

A questo punto è semplice dedurre il funzionamento della seconda sintassi, infatti, il **'foreach'** può estrarre sia la chiave che il valore degli elementi contenuti nell'array. Complicando ulteriormente lo script precedente si può verificare il funzionamento della seconda sintassi.

```

0  <?
1      foreach ($array_2 as $chiave => $valore) {
2          echo "<br>\$val = ".$valore;
3          if(is_integer($valore)){
4              echo " -----&gt; ed è un intero...";
5              echo " -----&gt; questa la sua chiave: ".$chiave;
6              // possono seguire istruzioni per trattare i dati interi
7          }else{
8              echo " -----&gt; ma non è un intero...";
9              echo " -----&gt; questa la sua chiave: ".$chiave;
10             // possono seguire istruzioni per trattare i dati NON interi
11         }
12     }
13 ?>

```

Sono state aggiunte le righe 5 e 9 che visualizzano il valore delle chiavi ed è stata modificata la sintassi della riga 1 in cui si chiede al **'foreach'** di estrarre anche le chiavi degli elementi dell'array oltre al valore e di assegnarli alle variabili **'\$chiave'** e **'\$valore'**. Il risultato è il seguente:

```

$val = 1 -----> ed è un intero... -----> questa la sua chiave: uno
$val = mela -----> ma non è un intero... -----> questa la sua chiave: frutta
$val = cinque -----> ma non è un intero... -----> questa la sua chiave: 5
$val = 30 -----> ed è un intero... -----> questa la sua chiave: 40

```

In conclusione il costrutto **'foreach'** è una struttura ciclica dedicata alla manipolazione degli array. Ovviamente possono essere gestiti anche array multidimensionale con dei **'foreach'** annidati, il lettore può approfondire l'argomento sul manuale ufficiale.

L'esempio trattato è raggiungibile all'indirizzo: <http://www.urcanet.it/brdp/php_manual/esempi/cap_2/foreach.php>

2.5.9 break

Il costrutto **'break'** serve ad abbandonare una struttura di controllo, nel dettaglio permette di uscire da: **'for'**, **'foreach'**, **'while'**, **'do..while'** e **'switch'**, e ammette un parametro numerico opzionale. Tale parametro deve essere un intero che indicherà il "livello" della struttura da abbandonare. Ad esempio, nel caso di due o più strutture annidate, si potrebbe voler uscire da una o più di esse: questo è possibile tramite il parametro.

Con l'occasione verranno approfondite, negli esempi seguenti, le strutture tralasciate in precedenza.

```

0  <?
1
2  $i = 0;
3  while (++$i) {    // il while incrementa anche in questo modo
4
5      if($i == 3){
6
7          echo "<br>Conto fino a ".$i." ed esco solo dal conteggio: ";
8
9          for ($t = 1; ;$t++) {    // la condizione di stop è tra le istruzioni.
10             if ($t > $i) {
11                 break 1;    // esce solo dal for equivale a break senza parametri.
12             }
13             echo " ".$t;
14         }
15     }

```

```

16     }elseif($i == 9){
17
18         echo "<br> Conto fino a ".$i." ed esco dal for e dal while:";
19
20         $t = 1;
21         for (;;) { // Nessuna condizione è espressa nel costrutto.
22             if ($t > $i) {
23                 break 2; // il parametro è 2 quindi esce dal for e dal while.
24             }
25             echo " ".$t;
26             $t++;
27         }
28
29     }
30
31 }
32
33 echo "<br><br> fine dello script!";
34
35 ?>

```

Come detto in precedenza, è importante curare l'indentazione del codice. L'ultimo esempio è composto da un ciclo **'while'** che include un **'elseif'** e due cicli **'for'** che, a loro volta, contengono un **'if'** ciascuno. Tutto questo susseguirsi di parentesi può creare confusione e rendere il codice illeggibile se non si presta attenzione all'indentazione.

Nella maggior parte dei casi, se il codice è ben scritto e ragionato, non si ha bisogno della struttura **'break'**. Tuttavia è bene sapere che esiste.

2.5.10 include()

Il PHP permette di includere uno o più file in un punto ben preciso di un altro file. Tali file possono, a loro volta, contenere codice PHP che verrà interpretato se delimitato dagli appositi marcatori.

Si pensi alla gestione di modelli grafici, alla necessità di dichiarare le stesse variabili in più file, all'inclusione di funzioni già pronte e via dicendo. In tutti questi casi è possibile creare un file specifico in cui inserire il codice che va incluso negli script. Ovviamente è possibile includere anche file residenti su macchine diverse dalla nostra. Di seguito vedremo come fare.

Per l'inclusione dei file il PHP offre diverse soluzioni. In questa guida verrà approfondita la funzione **'include()'**. Ne esistono altre che sono riportate nel manuale ufficiale.

La sintassi di base è la seguente:

```

0  <?
1      // inclusione di un file chiamato esterno.html
2
3      include('esterno.html');
4
5  ?>

```

è preferibile utilizzare le stringhe con double quote perchè meglio gestite e più versatili:

```

0  <?
1      // oppure con il double quote
2
3      include("esterno.html");
4
5  ?>

```

Come accennato in precedenza, il file da includere può trovarsi su una macchina diversa dalla nostra. Supponiamo che sia accessibile tramite il protocollo HTTP. La sintassi sarà:

```
0  <?
1  /* inclusione di un file remoto.
2      Su una macchina diversa, tramite
3      il protocollo HTTP
4  */
5
6  include("http://www.url_altro_sito.com/esterno.html");
7
8  ?>
```

Questo modo di operare è del tutto corretto. Tuttavia va tenuto conto dei seguenti fattori:

- l'elevato tempo di accesso al server remoto, che può causare un ritardo nella generazione della pagina e può rallentare notevolmente la visualizzazione della stessa;
- l'elaboratore server potrebbe essere irraggiungibile o il file essere inesistente;
- il contenuto del file remoto potrebbe essere modificato a nostra insaputa.

Alcuni di questi ostacoli possono essere aggirati, altri no. Ad esempio, il primo problema non è risolvibile. Per quanto una connessione sia veloce ed i servizi performanti si aggiunge sempre un intervallo di tempo dovuto alla procedura di richiesta del file. Questo è un parametro molto importante, che va tenuto presente quando si effettua una scelta di questo tipo. Il tempo necessario, inoltre, non è sempre lo stesso ma dipende da svariati fattori, ad esempio il traffico sul sito remoto, il traffico sulla rete, ecc...

Il secondo punto può essere affrontato in vari modi, quello più sbrigativo è sicuramente la gestione dell'errore generato con la mancata inclusione del file. Ecco un modo di procedere:

```
0  <?
1  /* inclusione di un file remoto. Su una macchina diversa,
2      tramite il protocollo HTTP.
3      Compresa la gestione dell'errore!!!
4  */
5
6  if( !@include("http://www.url_altro_sito.com/esterno.html") )
7      echo "<br><br> Problemi di visualizzazione! torna a trovarci...";
8
9  ?>
```

In questo modo si è reso invisibile l'eventuale messaggio di errore dell'interprete tramite il prefisso '@'. Per il resto si tratta di un semplice **'if'** che verifica il valore restituito dalla funzione **'include()'**, se il valore è falso visualizza il messaggio di errore, altrimenti include il file e prosegue.

Se il file remoto è uno script, ad esempio PHP, prima di essere incluso viene interpretato dal server, quindi viene incluso il risultato dello script generato in remoto e non sull'elaboratore su cui è ospitato il nostro script .

Passaggio di variabili tramite l'HTTP

Fino ad ora sono stati trattati i casi di singole pagine PHP in cui le variabili sono valorizzate, manipolate e visualizzate con l'interpretazione del singolo file. In questo modo l'utente che naviga la pagina non è in grado di interagire con gli script, non può passare dei dati al programma perchè non può editarne il codice. Il protocollo HTTP permette, tramite i marcatori HTML di passare dei dati tra le pagine, il PHP è in grado di ricevere ed elaborare queste informazioni.

Nelle pagine seguenti verranno trattati i metodi di scambio di variabili dando per scontata la conoscenza, come detto in principio, dell'HTML.

L'HTML permette al navigatore di inserire dati tramite il marcatore **'form'** che riceve dagli attributi **'action'** e **'method'** le direttive relative al file che deve ricevere i dati e al modo in cui essi devono essergli passati. L'attributo **'method'** ammette due possibili valori: **'get'** e **'post'**. Di seguito saranno approfondite le differenze tra i due metodi.

3.1 Metodo get

Quando si sceglie il metodo **'get'** le variabili ed il relativo valore vengono fornite allo script destinatario tramite la barra dell'indirizzo del *browser*.

Il modo migliore per chiarire il concetto è l'utilizzo di un esempio. Verranno realizzati due script, 'get_1.html' e 'get_2.php'. 'get_1.html' invierà i dati contenuti nel **'form'** a 'get_2.php' utilizzando il metodo **'get'**, 'get_2.php' riceverà i dati, li manipolerà e li visualizzerà in una semplice pagina di saluto. Questo l'indirizzo dell'esempio: http://www.urcanet.it/brdp/php_manual/esempi/cap_3/get_1.html

Ecco il codice del file 'get_1.html':

```

0      <html>
1      <head>
2          <title> Passaggio del nome! </title>
3      </head>
4
5      <body>
6
7      <br>
8          Una semplice pagina HTML che passa nome e cognome ad
9          uno script PHP che saluterà il navigatore.
10     <br><br>
11
12     <form method="get" action="get_2.php">
13
14         Dimmi il tuo nome: <input type="Text" name="nome"> <br><br>
15         Ed ora il Cognome: <input type="Text" name="cognome"> <br><br>
16
17         <input type="Submit" value="Adesso invia i dati in GET &gt;&gt;">
18
19     </form>
20
21 </body>
22 </html>
```

Il primo file invia i due campi di testo al file destinatario 'get_2.php' che riceve due variabili valorizzate con i dati inseriti dal navigatore. Le variabili hanno il nome del campo del **'form'**, nell'esempio le variabili sono **'\$nome'** e **'\$cognome'** e sono di tipo stringa.

Il codice del secondo script è molto semplice, non fa altro che visualizzare i valori delle due variabili.

```

0      <html>
1      <head>
2          <title> Pagina di destinazione... </title>
3      </head>
4
5      <body>
6
7          Ciao, <br><br>
8
9          Il tuo nome è: <?=$nome?> <br>
10         e il tuo cognome è: <?=$cognome?>
11
12     </body>
13 </html>

```

Figura 3.1. Come appare il **'form'** di invio dei dati.

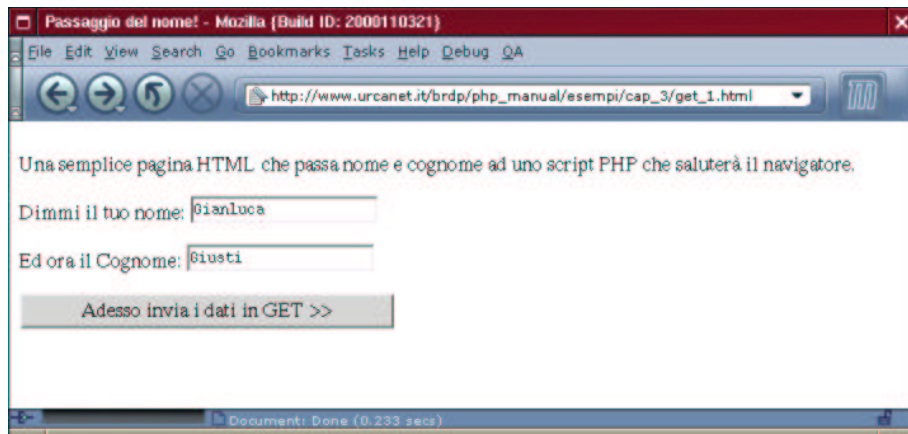
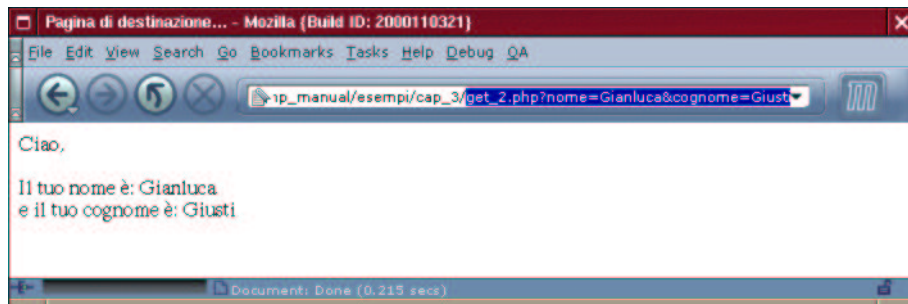


Figura 3.2. Nella barra degli indirizzi è evidenziata la sintassi del metodo **'get'** per il passaggio delle variabili.



L'obiettivo di questo esempio è quello di capire come funziona il metodo **'get'**, per farlo basta leggere il contenuto della barra degli indirizzi nel *browser* durante la visualizzazione dello script destinatario. L'indirizzo è simile a:

`http://www.urcanet.it/...ual/esempi/cap_3/get_2.php?nome=Gianluca&cognome=Giusti`

Risulta semplice e intuitivo capire come vengono passate variabili e valori. All'indirizzo della pagina viene aggiunto un carattere **'?'** e di seguito tutte le coppie di variabili e rispettivo valore separate dal **'='**, ad esempio: **'nome=Gianluca'**. Le coppie sono poi separate dal carattere **'&'**.

3.2 Metodo post

Nel metodo **'post'** i dati non vengono visualizzati in nessun modo, essi sono spediti tramite il protocollo HTTP e non sono visibili sul navigatore.

Una semplice verifica può essere eseguita modificando lo script `'get_1.html'` nel seguente modo:

```
0      <html>
1      <head>
2          <title> Passaggio del nome! </title>
3      </head>
4
5      <body>
6
7      <br>
8          Una semplice pagina HTML che passa nome e cognome ad
9          uno script PHP che saluterà il navigatore.
10     <br><br>
11
12     <form method="post" action="get_2.php">
13
14         Dimmi il tuo nome: <input type="Text" name="nome"> <br><br>
15         Ed ora il Cognome: <input type="Text" name="cognome"> <br><br>
16
17         <input type="Submit" value="Adesso invia i dati in GET &gt;&gt;">
18
19     </form>
20
21 </body>
22 </html>
```

Salvato come `'post_1.html'` è possibile testarne il funzionamento e verificare che nella pagina `'get_2.php'` la barra degli indirizzi del programma di navigazione è priva delle coppie di variabili, allo stesso tempo lo script ha funzionato e viene visualizzato il messaggio di saluto correttamente.

Analizzando lo script `'post_1.html'` si nota che l'unica modifica è quella all'attributo **'method'** del marcatore **'form'**.

3.3 Quale metodo scegliere?

Non c'è una regola particolare per scegliere l'uno piuttosto che l'altro metodo. Possiamo dire che se la quantità di dati da inviare è grande è preferibile utilizzare il **'post'**. Il protocollo HTTP permette, tramite i form HTML, l'upload di file dall'elaboratore cliente a quello server, in questi casi è bene utilizzare il metodo **'post'**.

Nel file di configurazione del PHP (`'php.ini'`) può essere definita sia la dimensione massima dei file che il server accetterà tramite HTTP, sia la dimensione massima dei dati accettati tramite **'post'**. Quest'ultimo valore può essere definito valorizzando la variabile **'post_max_size'** nel `'php.ini'`.

Quando i dati da scambiare sono pochi e non importa che risultino direttamente visibili si può utilizzare il metodo **'get'**, che è comodo anche per quelle situazioni in cui si utilizza spesso il tasto avanti e indietro del navigatore. Le variabili sono memorizzate insieme all'indirizzo e quindi ogni volta che si richiama una pagina ad essa vengono rifornite le variabili e i loro valori. Per rendersi conto di questo basta tenere d'occhio la barra degli indirizzi durante le ricerche su un qualunque motore di ricerca, il 99% di essi passano i dati tramite il metodo **'get'**. È altrettanto vero che i dati passati al motore per le ricerche sono decisamente pochi, di solito si tratta di una stringa con dei parametri che il motore aggiunge automaticamente.

3.4 Cosa è cambiato nel PHP versione 4.2

Dalle versioni del PHP 4.2 e successive la gestione delle variabili passate tramite moduli HTML è sensibilmente cambiata. Nel file di configurazione viene valorizzata a vero la variabile *track_vars*, in questo modo tutte le variabili presenti in un modulo ed inviate ad uno script PHP vengono inserite in un array associativo contenente il nome delle variabile e il relativo valore.

Ad esempio nel caso dello script 'get_2.php' visto nelle sezioni 3.1 e 3.2 si avrebbe la necessità di distinguere il metodo di invio. Più precisamente nel caso di un invio tramite metodo 'get' il file 'get_2.php' dovrebbe essere così modificato:

```
0 <html>
1 <head>
2   <title> Pagina di destinazione... </title>
3 </head>
4
5 <body>
6
7   Ciao, <br><br>
8
9   Il tuo nome è: <?=$HTTP_GET_VARS["nome"]?> <br>
10  e il tuo cognome è: <?=$HTTP_GET_VARS["cognome"]?>
11
12 </body>
13 </html>
```

Le differenze sono lampanti e riguardano le righe 9 e 10. L'interprete prende il valore ricevuto dalla variabile predefinita *\$HTTP_GET_VARS* che come detto in precedenza è un array associativo

Nel caso di un invio tramite metodo 'post' il programma andrebbe così modificato:

```
0 <html>
1 <head>
2   <title> Pagina di destinazione... </title>
3 </head>
4
5 <body>
6
7   Ciao, <br><br>
8
9   Il tuo nome è: <?=$HTTP_POST_VARS["nome"]?> <br>
10  e il tuo cognome è: <?=$HTTP_POST_VARS["cognome"]?>
11
12 </body>
13 </html>
```

Questo tipo di gestione delle variabili può sembrare prolissa ma evita confusioni all'interprete e costringe il programmatore a specificare la variabile su cui operare.

Ovviamente se si hanno pagine sviluppate secondo le vecchie specifiche basterà settare a falso la variabile *track_vars* nel file di configurazione del PHP, in questo modo tutto dovrebbe funzionare correttamente.

L'utilizzo delle funzioni in PHP

Una funzione può essere intesa come un "sotto-programma" a cui si possono fornire dei valori per ottenere una risposta da essi dipendente o meno.

Il PHP mette a disposizione degli sviluppatori una vasta gamma di funzioni predefinite, tuttavia in alcune situazioni è necessario eseguire operazioni particolari, in questi casi è possibile definire delle funzioni personali che si comportano secondo le proprie necessità.

Si prenda d'esempio lo script del paragrafo 2.3.5 in cui si visualizzava la data formattata in un modo particolare, è improponibile dover scrivere 38 righe di codice ogni volta che si desidera visualizzare la data all'interno di una pagina HTML in questo caso la soluzione migliore è realizzare una funzione dedicata che restituisce la data mediante una semplice "chiamata". Per evitare errori e ripetitive modifiche è consigliabile realizzare uno o più file in cui raccogliere le funzioni personali più utilizzate, una sorta di libreria, da premettere in ogni pagina tramite il costrutto `'include()'`.

4.1 Le funzioni predefinite

Come detto in principio non è volontà dell'autore riportare l'elenco delle funzioni PHP, esse sono catalogate ed abbondantemente illustrate nel manuale ufficiale reperibile al sito internet: [<http://www.php.net>](http://www.php.net).

Di seguito si vuole illustrare la struttura del manuale e il modo in cui esso va consultato per cercare, scegliere ed infine utilizzare le innumerevoli funzioni predefinite del linguaggio. Come sempre il modo migliore è l'utilizzo di un semplice esempio. Si supponga di voler realizzare un programma che visualizzi una porzione di una stringa. Bisogna come prima cosa individuare le funzioni che permettono di manipolare le stringhe.

Nella sezione "IV. Guida Funzioni" sono catalogate le funzioni messe a disposizione del programmatore, esse sono raggruppate per argomento. La sottosezione "LXXXIII. String functions" contiene le funzioni dedicate alla gestione delle stringhe. Come prima cosa si trova una descrizione della sezione e poi un lungo elenco, in ordine alfabetico, di funzioni con accanto una breve ma molto intuitiva descrizione. La funzione `'substr()'` potrebbe risolvere il problema sollevato nell'esempio.

Nel primo blocco della pagina descrittiva si trova il nome della funzione subito seguito dalle versioni dell'interprete PHP che la gestiscono e da una breve descrizione. Nel caso particolare:

```
substr
(PHP 3, PHP 4 >= 4.0.0)

substr -- Return part of a string
```

A seguire la descrizione approfondita che inizia con una delle parti più importanti della documentazione, infatti, in una sola riga viene spiegato l'intero funzionamento del comando. Nell'esempio proposto:

```
string substr (string string, int start [, int length])
```

Come detto questa è forse la parte più importante, la prima parola descrive il tipo di dato restituito dalla funzione, nel caso particolare `'string'`, dunque la funzione restituisce una stringa. Dopo il nome della funzione tra parentesi i valori che la funzione può accettare ed il relativo tipo di dato, va sottolineato che tra parentesi quadre vengono elencati i parametri non obbligatori e il relativo tipo di dato.

Nel particolare la funzione accetta un primo dato di tipo stringa, separato da virgola il numero del carattere di partenza di tipo intero (si parte sempre da 0 e non da 1) ed infine come dato opzionale la lunghezza, di tipo intero, della porzione di stringa da estrarre.

A questo schema seguono una descrizione approfondita ed una serie di esempi di funzionamento. Gli esempi se presenti sono molto utili ed autoesplicativi, inoltre trattano varie eccezioni ed eventuali casi particolari.

In fine ma non meno importante la sezione contenente le funzioni correlate o in qualche modo relazionate con quella in oggetto. Il manuale spesso suggerisce di consultare anche altre funzioni, in questo caso:

See also `strrchr()` and `ereg()`.

Spesso non esiste un unico modo per raggiungere la soluzione del problema e spesso questa ultima sezione fornisce degli spunti molto interessanti che possono portare ad una soluzione più semplice e brillante, il consiglio è di non sottovalutarla.

A questo punto è doveroso trattare almeno un esempio di utilizzo della funzione scelta come campione.

```

0  <?
1      // La stringa iniziale va in $var
2      $var = "Questa è la stringa di partenza";
3      //      0123456789012345678901234567890
4      //      1234567890
5
6      // Le due righe superiori semplificano il conteggio dei caratteri.
7
8      // Questo esempio prende la parte: "la stringa"
9      $sub = substr($var, 9, 10);
10     echo $sub;
11
12     // Per selezionare gli ultimi 2 caratteri basta usare:
13     $sub = substr($var, -2);
14     echo "<br><br> ".$sub;
15
16     // Per selezionare i primi 2 caratteri basta usare:
17     $sub = substr($var, 0, 2);
18     echo "<br><br> ".$sub;
19
20     // Se non specifico la lunghezza della porzione continua fino alla fine
21     $sub = substr($var, 5);
22     echo "<br><br> ".$sub;
23  ?>

```

Il risultato di questo semplice script è simile a:

la stringa

za

Qu

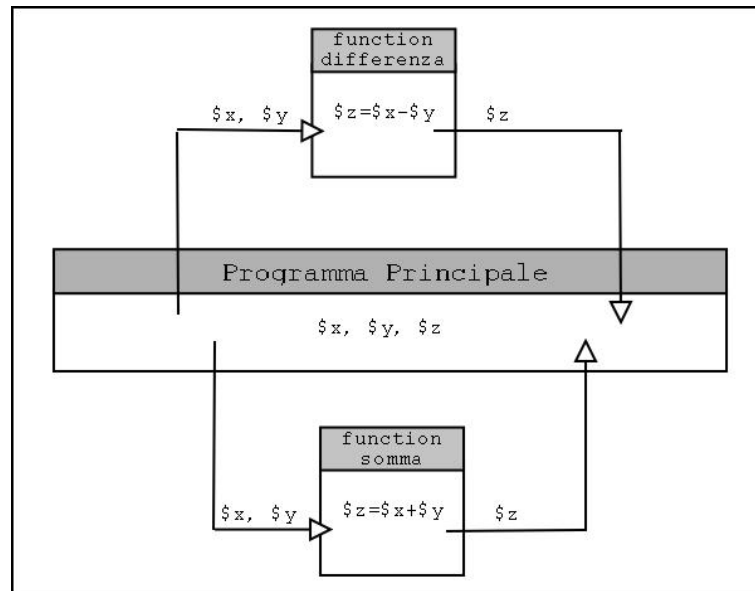
a è la stringa di partenza

Anche questo esempio è raggiungibile all'indirizzo: http://www.urcanet.it/brdp/php_manual/esempi/cap_4/substr.php

4.2 Le funzioni definite dall'utente

Un modo molto semplice di schematizzare il concetto di funzione è pensare ad una scatola che permette tramite due fori di inserire ed estrarre dei valori. All'interno della scatola può essere utilizzato qualunque costrutto messo a disposizione dal linguaggio, ecco perchè si parla di "sotto-programmi". Lo schema di figura 4.1 può essere di aiuto.

Figura 4.1. Il blocco di programma principale contiene due scatole (funzioni) che ricevono due valori e restituiscono in un caso la somma, in un altro la differenza.



L'esempio classico è quello di una semplice funzione di saluto:

```

0  <?
1  //----- Definizione delle funzioni
2
3  function saluto($ora){
4
5      /*
6       Questa funzione genera un messaggio di saluto
7       in base al valore della variabile $ora passatagli
8       che contiene l'ora, con valore tra 0 e 24
9      */
10
11     if (5 < $ora && $ora <= 12){
12         echo "Buon Giorno.";
13     }elseif(12<$ora && $ora <= 18){
14         echo "Buon Pomeriggio.";
15     }elseif(18<$ora && $ora <= 24){
16         echo "Buona Sera!";
17     }else{
18         echo "Guarda che razza di ora è! Ma quando dormi?!";
19     }
20 }
21
22
23 //----- Programma principale
24
25 $orario = date("H"); // estraggo l'ora attuale
26
27 // Richiamo la funzione e ci aggiungo del testo...
28
29 saluto($orario);
30 echo " Come va oggi?!?";

```

```

31
32 // non serve riscrivere il codice posso inserire
33 // il saluto con una sola RIGA di codice!!!
34
35 echo "<br><br> Ancora un saluto: "; saluto($orario);
36 ?>

```

Lo script è costituito da una parte contenente le dichiarazioni delle varie funzioni, nel caso specifico una sola, e dal programma principale che le richiama una o più volte. Come prima cosa è importante eseminare la sintassi per la dichiarazione delle funzioni.

```

function nome_funzione(variabile_1, variabile_2, .... , variabile_n){

    istruzioni funzione

    return valore se necessario

}

```

Il costrutto **'function'** segna l'inizio della dichiarazione, subito dopo va assegnato un nome che deve essere unico nel programma, non possono esserci più funzioni aventi lo stesso nome, tra parentesi tonde è possibile elencare le variabili, se esistono, che devono essere fornite alla funzione, se sono più di una vanno separate da una virgola (','). Il corpo della funzione è, infine, contenuto tra parentesi graffe ('{...}').

Una volta dichiarata, la funzione, può essere richiamata tutte le volte che si vuole, è questo il vantaggio che si ha nel suddividere il lavoro in piccoli problemi e a risolverli con piccoli e semplici sotto-programmi da richiamare ogni volta che se ne ha bisogno.

Come detto la funzione può restituire dei valori, dunque quando richiamata può essere vista come una variabile contenente dei valori. Se si modifica l'esempio precedente come riportato di seguito, la funzione non scriverà nulla sullo schermo ma valorizzerà tramite il comando **'return'** il nome della funzione, esso potrà essere visualizzato, assegnato ad un'altra variabile, ecc... Proprio come fosse una variabile.

```

0 <?
1 //----- Definizione delle funzioni
2
3 function saluto($ora){
4
5     /*
6      Questa funzione genera un messaggio di saluto
7      in base al valore della variabile ricevuta $ora
8      che contiene l'ora con valore tra 0 e 24
9      e lo restituisce al programma principale tramite
10     l'istruzione "return"
11     */
12
13     if (5 < $ora && $ora <= 12){
14         return "Buon Giorno.";
15     }elseif(12<$ora && $ora <= 18){
16         return "Buon Pomeriggio.";
17     }elseif(18<$ora && $ora <= 24){
18         return "Buona Sera!";
19     }else{
20         return "Guarda che razza di ora è! Ma quando dormi?!";
21     }
22 }
23
24 //----- Programma principale
25
26 $orario = date("H");
27
28 // Richiamo la funzione e ci aggiungo del testo...

```



```

29
30 $msg = saluto($orario);          // assegno il valore della funzione
31 echo $msg." Come va oggi?!?";
32
33 // una sola RIGA di codice CONCATENATO come fosse una variabile
34
35 echo "<br><br> Ancora un saluto: ".saluto($orario);
36
37 ?>

```

Le modifiche adottate nelle righe 14, 16, 18, 20 provvedono tramite il comando **'return'** ad assegnare un valore al nome della funzione invece di visualizzarlo sullo schermo. Di conseguenza cambia il modo di manipolare la funzione al momento della chiamata, infatti, in questo modo ritorna un valore che può essere trattato come una semplice variabile. Nella riga 30 viene assegnato il valore restituito dalla funzione ad un'altra variabile, mentre nella riga 35 viene concatenato in una stringa.

A questo punto si potrebbe avere la necessità di scegliere se far visualizzare il messaggio direttamente dalla funzione o farlo ritornare come valore. Complicando leggermente l'esempio si può risolvere il problema. Una semplice soluzione è quella di passare una seconda variabile, che a seconda del suo stato (vero o falso) fa comportare la funzione in due modi diversi.

```

0  <?
1  //----- Definizione delle funzioni
2
3  function saluto($ora, $modo){
4
5      /*
6          Questa funzione genera un messaggio di saluto
7          in base al valore della variabile ricevuta $ora
8          che contiene l'ora con valore tra 0 e 24
9          e lo restituisce al programma principale tramite
10         l'istruzione "return" se $modo è falso, se è vero
11         ne visualizza il valore.
12      */
13
14     if (5 < $ora && $ora <= 12){
15         $msg = "Buon Giorno.";
16     }elseif(12<$ora && $ora <= 18){
17         $msg = "Buon Pomeriggio.";
18     }elseif(18<$ora && $ora <= 24){
19         $msg = "Buona Sera!";
20     }else{
21         $msg = "Guarda che razza di ora è! Ma quando dormi?!";
22     }
23
24     // controllo sul flag $modo.
25     if($modo){
26         echo $msg;
27     }else{
28         return $msg;
29     }
30 }
31
32 //----- Programma principale
33
34 $orario = date("H");
35 $modo = false;
36
37 // Richiamo la funzione e ci aggiungo del testo...
38 $msg = saluto($orario, $modo);
39 echo $msg." Come va oggi?!?".<br><br>;
40
41
42 // Adesso posso anche farlo scrivere direttamente, ecco come:

```

```
43  saluto($orario,1);
44  ?>
```

Adesso la funzione si aspetta due parametri, il primo serve a generare il messaggio di saluto, il secondo (righe 25-29) a determinare il comportamento della funzione. Se cambia la dichiarazione deve cambiare anche la chiamata alla funzione, infatti, nelle righe 39 e 43 vengono passati i due parametri e la funzione si comporta di conseguenza.

Quando si richiama una funzione non importa il nome delle variabili che gli vengono fornite, quello che importa è l'ordine con cui esse vengono elencate.

Dunque i valori passati alla funzione andranno nelle variabili locali (della funzione) definite al momento della dichiarazione della stessa. Nell'esempio seguente viene approfondito questo aspetto tramite una funzione che genera e visualizza una scheda informativa di un potenziale utente.

```
0  <?
1  //----- Definizione delle funzioni
2
3  function scheda($nome, $cognome){
4
5      /*
6          Questa funzione ha lo scopo di chiarire il concetto di
7          variabile locale e globale. Fare inoltre attenzione
8          all'ordine con cui sono passate le variabili
9      */
10
11     global $tel;
12
13     echo "<br>Scheda informativa utente: <br><br> ";
14     echo "<br>Nome: ".$nome;
15     echo "<br>Cognome: ".$cognome;
16     echo "<br>Telefono: ".$tel;
17     echo "<br>e-mail: ".$email;
18     echo "<br><br>-----<br>";
19 }
20
21 //----- Programma principale
22
23 $nome = "Gianluca";
24 $cognome = "Giusti";
25 $tel = "06/66666666";
26 $email = "brdp @ urcanet.it";
27
28 // richiamo la funzione che genera la scheda.
29 scheda($nome, $cognome);
30
31 // il nome delle variabili non importa! Importa invece l'ordine
32 // con cui vengono passate alla funzione! Ecco la prova:
33 scheda($cognome, $nome);
34 ?>
```

Come detto in principio, le funzioni sono dei piccoli programmi a se stanti, dunque le variabili utilizzate al loro interno, dette "variabili locali", non hanno lo stesso valore di quelle utilizzate nel programma principale, dette "variabili globali" anche se nominate allo stesso modo. Si può dire che sono variabili diverse. Un modo per far coincidere le variabili locali e globali è passare le variabili globali come argomento della funzione, proprio come fatto negli esempi precedenti. Un secondo metodo, è l'utilizzo del comando **'global'**. Nella riga 11 dell'esempio precedente viene utilizzato per "catturare" il valore della variabile globale *\$tel* all'interno della funzione.

Il risultato dell'ultimo programma è simile a:

Scheda informativa utente:

Nome: Gianluca
Cognome: Giusti
Telefono: 06/66666666
e-mail:

Scheda informativa utente:

Nome: Giusti
Cognome: Gianluca
Telefono: 06/66666666
e-mail:

Come prima cosa va notata l'assenza dell'indirizzo e-mail, questo dipende dal fatto che la variabile *\$email* non è stata fornita in alcun modo alla funzione quindi al suo interno non è valorizzata. Al contrario le altre variabili sono valorizzate, *\$nome* e *\$cognome* vengono passate al momento della chiamata mentre la variabile *\$tel* viene associata a quella globale.

L'utilizzo di **'global'** non è consigliabile dal momento che ogni modifica alla variabile utilizzata come globale all'interno della funzione si ripercuote anche nel programma principale. Se non si hanno motivi particolari non ha senso utilizzare questo metodo.

Un ottimo esercizio per l'utente è la verifica di quanto appena detto. Modificare l'ultimo esempio al fine di dimostrare che le modifiche alla variabile *\$tel* effettuate all'interno della funzione si ripercuotono anche nel programma principale, mentre eventuali modifiche alle variabili passate come argomento all'interno della funzione non hanno effetto nel programma principale.

Per i più esperti la situazione è simile al passaggio di variabili per "valore" o per "riferimento".

La funzione, inoltre, viene richiamata due volte invertendo l'ordine dei parametri a dimostrazione che il nome delle variabili non ha alcun importanza ma quello che importa è l'ordine con cui vengono dichiarate e passate al momento della chiamata.

In conclusione il modo migliore e più elegante per realizzare la scheda utente dell'esempio è fornire tutti i dati necessari alla funzione come parametri della chiamata.

```

0  <?
1  //----- Definizione delle funzioni
2
3  function scheda($nome, $cognome, $tel, $email){
4
5      // Il modo più corretto è il passaggio delle variabili
6      // Salvo casi particolari in cui si è obbligati ad usare il global
7
8      echo "<br>Scheda informativa utente: <br><br> ";
9      echo "<br>Nome: ".$nome;
10     echo "<br>Cognome: ".$cognome;
11     echo "<br>Telefono: ".$tel;
12     echo "<br>e-mail: ".$email;
13     echo "<br><br>-----<br>";
14 }
15
16
17 //----- Programma principale
18
19 $nome = "Gianluca";
20 $cognome = "Giusti";
21 $telefono = "06/66666666";
22 $posta_elettronica = "brdp @ urcanet.it";
23

```

```

24 // richiamo la funzione che genera la scheda. I nomi non contano!
25 scheda($nome, $cognome, $telefono, $posta_elettronica);
26
27 ?>

```

Uno degli aspetti più importanti di un sito Internet è l'estetica ed è anche la parte che viene rinnovata più di frequente. Si pensi alla necessità di elencare degli oggetti secondo particolari vesti grafiche, utilizzando una funzione è possibile modificare varie pagine, in cui essa viene richiamata, modificando una sola porzione di codice. Un altro esempio è dato dai siti in cui è possibile personalizzare la formattazione dei dati tramite la scelta di modelli predefiniti, in questo caso la funzione potrebbe ricevere un parametro in più che rappresente il modello, oppure si potrebbero implementare varie funzioni che formattano i medesimi dati in modo differente.

Come conclusione si può senza dubbio affermare che un programma strutturato è molto più leggibile ed elegante di uno costituito da un unico blocco di codice spesso incomprensibile.

In fine va notato che ci sono delle operazioni che vengono ripetute spesso nel lavoro di tutti i giorni, ad esempio formattazione della data in vari formati, connessione e interrogazioni di basi di dati, paginazione di elenchi, gestione di immagini, upload di file, ecc... La cosa più intelligente è creare dei file, paragonabili a librerie, contenenti le dichiarazioni delle funzioni più utilizzate, a questo punto basterà includere tali "librerie" in cima agli script in modo da poter richiamare le funzioni senza dovrele dichiarare ogni volta. Ancora più corretto è la creazione di "librerie" contenenti delle classi con cui poter dichiarare dei veri e propri oggetti ma questo verrà spiegato nella sezione dedicata alla programmazione avanzata.

Un esercizio interessante lasciato al lettore è quello di implementare una funzione chiamata `'isint($var)'`, supponendo che non esista la funzione già vista `'is_integer()'`, che utilizzando `'gettype()'` restituisca un valore vero se la variabile passata è un intero o falso se la variabile è di un altro tipo.

4.3 Le funzioni ricorsive

Il PHP permette di richiamare le funzioni in modo ricorsivo, ovvero è possibile richiamare una funzione dal suo interno. Un classico esempio applicativo di questa tecnica è il calcolo del fattoriale. Come noto il fattoriale di un numero intero positivo n (indicato come $n!$) è pari a $n*(n-1)*(n-2)*...*(n-(n-1))$. Ad esempio:

$$5! = 5*4*3*2*1 = 120$$

Ecco come risolvere il problema con l'utilizzo di una funzione ricorsiva:

```

0<html><body>
1  <?
2    //----- Definizione delle funzioni
3
4    function fattoriale($numero){
5
6        if($numero <= 1){
7            return 1;    // abbandona la ricorsione
8        }else{
9            return $numero*fattoriale($numero-1);    //la funzione richiama se stessa
10       }
11
12   }
13
14
15   //----- Programma principale
16
17   echo "<br>Calcolo del fattoriale:<br>";
18

```

```
19  for($i=-2; $i<=10; $i++){
20      echo "<br>Il fattoriale di $i è: ".fattoriale($i);
21  }
22
23  ?>
24
25  </body></html>
```

All'interno della dichiarazione, riga 9, viene richiamata la funzione stessa passandogli il numero decrementato di uno. Questo perchè $n! = n*(n-1)! = n*(n-1)*(n-2)!$ e così via... Il fattoriale di 0 è pari ad 1 per definizione. Per semplificare il controllo si è stabilito che il fattoriale di un numero negativo è pari ad 1 anche se non è corretto matematicamente.

L'attenzione non deve essere focalizzata sull'aspetto matematico ma sul fatto che la funzione nel programma principale viene richiamata una sola volta. Ancora più importante in problemi di questo genere è la definizione di una condizione di arresto, in questo caso quando il numero passato alla funzione è ' ≤ 1 ' la funzione abbandona la ricorsione e restituisce valore pari ad 1. Se si sbaglia o si trascuria la condizione di uscita il programma potrebbe entrare in un ciclo infinito e non smetterebbe mai l'esecuzione.

La gestione del file system

Il file system è ben gestito dal PHP, le funzioni a disposizione degli sviluppatori sono innumerevoli, inoltre sono ben integrate con il sistema operativo in modo particolare GNU/Linux.

Ad esempio è possibile ottenere e cambiare i permessi, il proprietario e il gruppo di un file, creare collegamenti simbolici, copiare, spostare, eliminare directory e file. Dunque gli strumenti a disposizione non mancano e molti di essi verranno trattati nelle pagine seguenti.

Prima di entrare nello specifico è bene precisare ancora una volta che il file system su cui il PHP opera è quello dell'elaboratore servente, quindi se si crea un file esso viene creato sul disco del servente e non su quello del cliente.

Tramite il protocollo HTTP il PHP può gestire anche il trasferimento di file dall'elaboratore cliente a quello servente, questo però verrà trattato nella sezione dedicata alla programmazione avanzata.

5.1 Concetto di file

Un file può essere visto come un contenitore nel quale immagazzinare informazioni. La gestione fisica del file su disco viene effettuata dal sistema operativo a cui si interfaccia il PHP.

Per poter accedere alle informazioni contenute in un file è necessario aprirlo, leggerne il contenuto, eventualmente modificarlo e infine chiuderlo.

I dati vengono salvati nel file sotto forma di righe, per indicare il fine riga nei file di testo viene utilizzato '\n', mentre la fine del file è ottenuta tramite la funzione `feof()` che restituisce un valore booleano vero se ci si trova alla fine del file.

La particolarità fondamentale di questa struttura è il suo accesso, a differenza degli array non è possibile puntare direttamente ad una riga ben precisa del file è dunque necessario scorrere il suo contenuto fino ad arrivare alla riga desiderata. Questo tipo di accesso viene detto "accesso sequenziale".

Non ci sono, inoltre, limiti massimi o minimi per le dimensioni del file, esse dipendono dalla disponibilità di spazio su disco.

5.2 Apertura di un file

Se un file è un contenitore per poter accedere al suo contenuto è necessario aprirlo, per farlo si utilizza la funzione `fopen()` a cui va fornito il file con il percorso, se diverso da quello dello script, e il tipo di apertura. Dunque al momento dell'apertura del file bisogna dichiarare il tipo di operazione che si desidera eseguire su di esso, ad esempio se si vuole solo leggere, scrivere o semplicemente aggiungere dati alla fine. Ecco l'elenco dei possibili modi di accedere ad un file in PHP:

- `'r'` Apre in sola lettura; posiziona il suo puntatore all'inizio del file;
- `'r+'` Apre in lettura e scrittura; posiziona il suo puntatore all'inizio del file;
- `'w'` Apre in sola scrittura; posiziona il suo puntatore all'inizio del file e ne elimina il contenuto. Se il file non esiste lo crea;
- `'w+'` Apre in lettura e scrittura; posiziona il suo puntatore all'inizio del file e ne elimina il contenuto. Se il file non esiste lo crea;

- `'a'` Apre in sola scrittura; posiziona il suo puntatore alla fine del file. Se il file non esiste lo crea;
- `'a+'` Apre in lettura e scrittura; posiziona il suo puntatore alla fine del file. Se il file non esiste lo crea;

Il PHP permette anche di aprire file remoti tramite i protocolli più diffusi come HTTP e FTP, ovviamente il file deve essere raggiungibile almeno in lettura.

Ecco qualche esempio di apertura di file:

```
0 $f = fopen ("dati.txt", "w");
1 $f = fopen ("/home/qualche_percorso/dati.txt", "a");
2 $f = fopen ("http://www.php.net/", "r");
3 $f = fopen ("ftp://nome_utente:password@indirizzo_del_sito_ftp.com/", "w");
```

Va notato che la funzione ritorna un intero. La variabile `$f` contiene i riferimenti al file aperto, viene anche detto "puntatore al file" anche se non ha nulla a che vedere con la "struttura puntatore". Tutte le operazioni saranno eseguite sul puntatore al file.

Nella riga 0 il file viene aperto in sola scrittura un file che si trova nella stessa directory dello script, se non esistere viene creato. Nella riga 1 il file viene aperto in un ben preciso punto del disco del server. Negli ultimi due esempi i file aperti risiedono in remoto e sono aperti tramite i due protocolli HTTP ed FTP

Fare molta attenzione ai permessi dei file che si intende manipolare e delle directory in cui sono contenuti. Ricordare che l'utente che accede al file è quello che lancia il server HTTP e di conseguenza l'interprete PHP in esso caricato come modulo. Tale utente viene specificato nel file di configurazione del server.

5.3 Lettura di un file di testo

Il primo passo sarà quello di leggere un semplice file di testo esistente e contenuto nella stessa directory dello script. Verrà utilizzato il metodo di sola lettura con l'indispensabile controllo di errore e ancora prima di esistenza del file da leggere. Se si salvano informazioni riservate in file di testo è bene farlo in una directory che non faccia parte dei documenti WEB altrimenti potrebbe essere richiamato e letto direttamente dal navigatore. Inoltre è importante nascondere eventuali errori mediante una corretta gestione degli stessi al fine di evitare la visualizzazione di informazioni importanti come il percorso e il nome del file che si tenta di aprire. Prima di scrivere il codice è bene creare il file da leggere, nominarlo come `'dati_1.txt'` e riempirlo con del semplice testo.

Un esempio di `dati_1.txt`:

```
Salve a tutti,
scrivo nel file per poi leggere con php.
di seguito una riga vuota
```

e poi il file finisce!

Ecco il programma che legge il file appena creato:

```
0 <html><body>
1
2 <?
3 if (file_exists("dati_1.txt")){
4     echo "Il file dati_1.txt esiste. Ecco il suo contenuto:<br>";
5
```

```

6  $f = @fopen("dati_1.txt", "r"); // apre il file in sola lettura
7  if($f){
8      while(!feof($f)){ // un semplice while fino alla fine del file
9          $riga = fgets($f,4096); // legge la riga
10         echo "<br>".$riga; // visualizza la riga
11     }
12     @fclose($f); // è importante chiudere il file
13 }else{
14     echo "Errore durante l'apertura del file!";
15 }
16 }
17 }else{
18     echo "Il file dati_1.txt NON esiste!";
19 }
20 }
21 ?>
22
23 </body></html>

```

Nella riga 3 viene controllata l'esistenza del file di testo tramite la funzione `'file_exists()'` che come riportato nel manuale ufficiale alla sezione "XXVI. Filesystem functions" ritorna un valore booleano vero se il file esiste falso negli altri casi. Questa sezione del manuale è ricca di funzioni per la gestione dei file si consiglia vivamente di approfondirne la lettura.

Alla riga 6 il file viene aperto in sola lettura è bene fare attenzione alla gestione dell'errore. Se l'apertura va a buon fine tramite una struttura ciclica ne viene visualizzato il contenuto. La lettura viene interrotta nel momento in cui la funzione `'feof()'` restituisce valore vero, questo accade quando si arriva alla fine del file.

Per la lettura delle singole righe del file è stata utilizzata la funzione `'fgets()'` che restituisce una stringa o un valore falso in caso di errore.

Molto importante è l'istruzione della riga 12 con la quale viene chiuso il puntatore al file definito nella fase di apertura. È bene liberare un file prima possibile per evitare problemi di scrittura concorrente. Anche questa funzione ritorna un valore booleano vero se l'operazione va a buon fine, falso altrimenti.

Se il file di testo da leggere è piccolo si può utilizzare la funzione `'file("nome_del_file")'`, ovviamente oltre al nome del file può essere specificato un particolare percorso e il tutto può essere contenuto in una variabile da passare come parametro. La funzione `'file()'` restituisce un array formato da tanti elementi quante sono le righe del file di testo, ogni elemento conterrà il testo della riga relativa. Ad esempio nel caso del precedente file `'dati_1.txt'`:

```

0  <html><body>
1
2  <?
3  $appo = @file("dati_1.txt");
4  if ($appo){
5      echo "Operazione riuscita il contenuto del file dati_1.txt
6          è nell'array <b>\$appo</b><br>";
7          $i=0;
8          foreach($appo as $valore) // visualizzo il contenuto dell'array
9              echo "<br>".$valore;
10
11 }else{
12     echo "Errore nella lettura del file! Accertarsi che
13         esista e che i permessi siano corretti";
14 }
15
16 ?>
17
18 </body></html>

```

Questa funzione risulta molto comoda e rapida in quanto non necessita di istruzioni per l'apertura e la chiusura del file. È evidente che in questo modo il contenuto può essere soltanto letto e non

modificato. In caso di file di dimensioni medio grandi è preferibile utilizzare le funzioni classiche viste in precedenza che benchè più macchinose garantiscono una migliore gestione del file e della memoria.

5.4 Scrittura in un file di testo

Come al solito prima di poter operare sul contenuto del file bisogna aprirlo, ovviamente per poter inserire dei dati nel contenitore bisogna aprirlo in modalità di scrittura. Come visto in precedenza esistono varie modalità e per ogni situazione va scelta quella ottimale. Di seguito esamineremo alcuni metodi di scrittura nel file.

Il tipo `'r+'` permette di leggere e scrivere nel file che si sta aprendo e posiziona il puntatore all'inizio del contenitore senza alterarne a priori il contenuto. È utile quando bisogna posizionare una riga in una ben precisa posizione del file.

Nel caso di `'w'` e `'w+'` il file viene aperto in scrittura o scrittura e lettura e viene immediatamente svuotato. Può essere comodo nel caso di file di appoggio che possono essere sovrascritti senza doverne controllare il contenuto.

Il metodo di apertura `'a'` e `'a+'`, che serve ad accodare righe al file, verrà trattato in dettaglio nel paragrafo 5.5.

Nell'esempio seguente si salva nel file `'dati_3.txt'` l'indirizzo IP dell'ultimo visitatore che ha caricato la pagina `'3.php'`. Anche questo come tutti gli esempi di questo capitolo è raggiungibile al solito indirizzo: `<http://www.urcanet.it/brdp/php_manual/esempi/>` nel caso specifico: `<http://www.urcanet.it/brdp/php_manual/esempi/cap_5/3.php>`

```

0  <html><body>
1
2  <?
3      $f = @fopen("dati_3.txt", "w");    // lo apre e lo svuota. se non esiste lo crea
4      if($f){
5          echo "<br><br>file <a href=\"dati_3.txt\">dati_3.txt</a> Aperto correttamente.";
6          echo "<br><br>Sto salvando i tuoi dati nel file";
7
8          $frase = "Ultimo visitatore ad aver eseguito lo script 3.php \n\n";
9          $frase = $frase."il suo IP: ".$REMOTE_ADDR." \n";
10
11         @fputs($f,$frase);    // scrive la frase nel file tramite $f
12         @fclose($f);         // è importante chiudere il file
13
14         echo "..... Fatto!";
15
16     }else{
17         echo "<br>Errore durante l'apertura del file dati_3.txt";
18     }
19 ?>
20
21 </body></html>

```

Nella riga 3 il file viene aperto in sola scrittura e come più volte ripetuto il suo contenuto viene cancellato. Dunque questo codice sarà in grado di mantenere traccia soltanto dell'ultimo visitatore. Sarebbe più interessante memorizzare più visite ma bisogna fare i conti con la crescita del file, se la pagina è molto frequentata le dimensioni del file di log potrebbero crescere a dismisura fino a saturare lo spazio su disco. Un buon esercizio potrebbe essere la realizzazione di uno script che tenga traccia degli ultimi `'$n'` visitatori, dove `'$n'` è un parametro che può essere impostato a seconda delle esigenze.

Di seguito una possibile soluzione:

```

0 <html><body>
1 <?
2 $n = 4; // numero di visite da mantenere
3 $f = @fopen("dati_4.txt", "r");
4
5 if($f){
6
7     echo "<br>file dati_4.txt esistente ed aperto correttamente <br>";
8     $i=0;
9     while( !feof($f) && $i<$n-1 ){ // finchè fine del file o il $i< $n-1
10         $righe[$i] = @fgets($f,4096); // file piccolo! utilizzo array sacmbio
11         $i++;
12     }
13     @fclose($f); // è importante chiudere il file
14
15     $f = @fopen("dati_4.txt", "w"); // adesso lo riempio con i nuovi dati.
16     if($f){
17         echo "<br><br>file <a href=\"dati_4.txt\">dati_4.txt</a> Aperto
18             correttamente. Sto salvando i tuoi dati";
19
20         $frase = "visitatore di 4.php - ";
21         $frase = $frase."il suo IP:".$$REMOTE_ADDR." alle ore: ".date("d-m-Y G:i:s")." \n";
22         @fputs($f,$frase); // scrive la frase nel file tramite $f
23
24         for ($i=0; $i<count($righe); $i++)
25             @fputs($f,$righe[$i]);
26
27         @fclose($f); // è importante chiudere il file
28         echo ".....Fatto!";
29     }
30 }else{
31
32     echo "<br>Il file dati_4.txt non esiste. Lo creo.";
33
34     $f = @fopen("dati_4.txt", "w");
35
36     if($f){
37
38         echo "<br><br>file <a href=\"dati_4.txt\">dati_4.txt</a> Aperto
39             correttamente. Sto salvando i tuoi dati";
40
41         $frase = "visitatore di 4.php - ";
42         $frase = $frase."il suo IP:".$$REMOTE_ADDR." alle ore: ".date("d-m-Y G:i:s")." \n";
43
44         @fputs($f,$frase); // scrive la frase nel file tramite $f
45         @fclose($f); // è importante chiudere il file
46
47         echo ".....Fatto!";
48     }else{
49         echo "<br><br>Non posso creare il file dati_4.txt";
50     }
51 }
52 }
53 }
54 ?>
55 </body></html>

```

Ci sono vari modi per risolvere il problema, quello usato per questo esempio è volutamente prolisso al fine di utilizzare più volte le funzioni sotto esame così da rendere più semplice l'apprendimento.

Concettualmente si è ragionato in questi termini:

```

se il file esiste
{
    - leggo le prime $n-1 righe del file
    - le salvo in un array di appoggio

```

```

- chiudo il file
- apro il file in scrittura cancellandone il contenuto
- salvo i dati relativi alla connessione attuale nel file
- salvo tutte le righe dell'array di appoggio
- chiudo il file
}
altrimenti
{
- apro il file in scrittura e viene creato
- salvo i dati relativi alla connessione attuale nel file
- chiudo il file
}
}

```

in realtà non viene fatto un controllo sull'esistenza del file ma si è considerato che se non si riesce ad aprire in sola lettura con il metodo `'r'` il file o non esiste o non è leggibile.

Nel ciclo di riga 9 vengono prese le prime `'$n-1'` righe del file o tutto il file se ha meno di `'$n'` righe, la condizione che per prima non viene soddisfatta fa abbandonare il ciclo.

Nelle righe 21 e 43 è stata usata la variabile predefinita `'$REMOTE_ADDR'` che contiene l'indirizzo IP del visitatore e di seguito la funzione `'date()'` già incontrata nella sezione 1.1 che in questo caso ritorna la data nella forma del tipo: `'31-10-2002 15:13:13'`. Maggiori informazioni in merito possono essere trovate nella sezione "XVI. Date and Time functions" del manuale ufficiale.

In fine nella riga 24 al posto della solita struttura `'foreach'` è stata utilizzata volutamente la funzione `'count()'` che restituisce il numero di elementi dell'array che gli viene passato come parametro.

In conclusione va notato che l'ultima visita verrà registrata in testa al file mentre le più vecchie verranno scartate mano a mano che ne entreranno delle nuove. Il risultato ottenuto nel file `'dati_4.txt'` è simile a:

```

visitatore di 4.php - il suo IP:192.168.1.153 alle ore: 31-10-2002 15:13:24
visitatore di 4.php - il suo IP:192.168.1.153 alle ore: 31-10-2002 15:13:17
visitatore di 4.php - il suo IP:192.168.1.153 alle ore: 31-10-2002 15:13:12
visitatore di 4.php - il suo IP:192.168.1.153 alle ore: 31-10-2002 15:10:21

```

Come conclusione viene proposto un esempio riepilogativo in merito al quale non verranno fornite spiegazioni, sarà compito del lettore capirne il comportamento. Per testarne il funzionamento è tuttavia possibile collegarsi all'indirizzo: http://www.urcanet.it/brdp/php_manual/esempi/cap_5/5.php

```

0 <html><body>
1 <?
2 /*
3 Primo tentativo di apertura del file inesistente dati_5.txt in sola lettura
4 il tentativo di apertura fallisce perchè la modalità 'r' non
5 crea il file in caso di inesistenza, a differenza delle modalità
6 'w' e 'a' che se non esiste lo creano. Quindi il controllo seguente
7 restituirà il messaggio di errore...
8 */
9
10 $f = @fopen("dati_5.txt", "r");
11
12 if($f){
13     echo "<br>1) file dati_5.txt Aperto. Ecco il contenuto: <br>";
14     //Se apre il file facciamo scrivere il contenuto
15     while(!feof($f)){ // finchè non raggiungo la fine del file
16         $riga = @fgets($f,4096);
17         echo "<br>".$riga;
18     }
19     @fclose($f); // è importante chiudere il file
20 }else{
21     echo "<br>1) Apertura del file dati_5.txt FALLITA! Ora lo creo e lo riempio.";
22 }
23

```

```

24 /*
25 Secondo tentativo apro in sola scrittura il file
26 il tentativo di apertura riesce perchè la modalità 'w'
27 crea il file in caso di inesistenza. Il controllo seguente
28 restituirà messaggio positivo.
29 ATTENZIONE in questo caso il file viene svuotato al momento
30 dell'apertura, il contenuto viene perso.
31 */
32
33 if (!file_exists("dati_5.txt")){
34
35     $f = @fopen("dati_5.txt", "w");
36     if($f){
37
38         echo "<br><br>2) file dati_5.txt Aperto correttamente. Lo riempio";
39
40         // sappiamo che va a buon fine e quindi scriviamoci qualcosa dentro.
41         // notare che \n inserisce un fine riga!!!
42         $frase = "Ciao Popolo. \n";
43         $frase = $frase."Scrivo nel mio primo file in PHP \n";
44         $frase = $frase."Speriamo bene! \n";
45         $frase = $frase."CIAO";
46
47         @fputs($f,$frase);    // scrive la frase nel file tramite $f
48         @fclose($f);        // è importante chiudere il file
49
50     }else{
51         echo "<br><br>2) Apertura del file dati_5.txt <strong>FALLITA!</strong>";
52     }
53
54 }else{
55
56     echo "<br><br>2) Il file dati_5.txt esiste già. Allora lo RIMUOVO!";
57     if(@unlink ("dati_5.txt")){
58         echo "<br><br> FILE dati_5.txt RIMOSSO CORRETTAMENTE";
59     }else{
60         echo "<br><br> ERRORE durante la rimozione di dati_5.txt";
61     }
62 }
63 ?>
64 </body></html>

```

In questo ultimo esempio viene utilizzata per la prima volta la funzione `'unlink()'` che serve ad eliminare il file che riceve come parametro, essa restituisce un intero che assume valore pari a 0 o falso se l'operazione di cancellazione non va a buon fine.

5.5 Accodare testo in un file

Ovviamente esistono metodi più semplici, immediati e dunque migliori per tenere traccia di qualunque evento. Prima di continuare è bene ricordare che il server HTTP memorizza in file di log le informazioni relative alle connessioni quindi se non si hanno particolari esigenze tracciare le visite degli utenti potrebbe essere un'operazione ridondante.

Il PHP come visto nella sezione 5.2 permette di aprire un file in modalità "append", ovvero in modo che le righe inserite vengano appese alla fine del file. questo metodo è utilizzabile passando alla funzione `'fopen()'` il parametro `'a'` o `'a+'`, nel primo caso il file viene aperto in sola scrittura mentre nel secondo in scrittura e lettura, il puntatore viene posizionato alla fine del file e se non esiste il file specificato viene creato.

Se si pensa al problema affrontato precedentemente (sezione 5.4) ci si rende immediatamente conto che utilizzando questo metodo di scrittura nel file si possono accodare le informazioni utilizzando poche righe di codice. Ad esempio:

```

0  <html><body>
1
2  <?
3      $f = @fopen("dati_6.txt", "a");    //apre in scrittura. se non esiste lo crea
4      if($f){
5          echo "<br><br>file <a href=\"dati_6.txt\">dati_6.txt</a> Aperto.";
6          echo "<br><br>Sto salvando i tuoi dati nel file";
7
8          $frase = "visitatore di 6.php - ";
9          $frase = $frase."il suo IP:".$$REMOTE_ADDR." alle ore: ".date("d-m-Y G:i:s")." \n";
10
11         @fputs($f,$frase);    // scrive la frase nel file tramite $f
12         @fclose($f);          // è importante chiudere il file
13
14         echo "..... Fatto!";
15
16     }else{
17         echo "<br>Errore durante l'apertura del file dati_6.txt";
18     }
19 ?>
20
21 </body></html>

```

È bene sottolineare che in questo caso non ci sono controlli sulle dimensioni del file di testo generato. Ogni volta che lo script viene eseguito il file 'dati_6.txt' aumenta di dimensioni. Se la pagina è molto visitata a lungo andare potrebbe addirittura saturare il disco. Visto che questo è un semplice esercizio e non si ha alcun bisogno dei dati contenuti nel file verrà aggiunto un controllo che elimina il file se le sue dimensioni superano un determinato valore. Ecco una soluzione:

```

0  <html><body>
1
2  <?
3      $max_file_size = 4000;    // dimensione massima in byte
4
5      if(filesize("dati_6.txt") > $max_file_size ){
6          echo "<br><br><b>Vuoto il file! ha superato il limite</b><br><br>";
7          $f = @fopen("dati_6.txt", "w");    //lo apre scrittura e lo svuota
8      }else{
9          $f = @fopen("dati_6.txt", "a");    //lo apre scrittura e appende i dati
10     }
11     if($f){
12         echo "<br><br>file <a href=\"dati_6.txt\">dati_6.txt</a> Aperto.";
13         echo "<br><br>Sto salvando i tuoi dati nel file";
14
15         $frase = "visitatore di 6.php - ";
16         $frase = $frase."il suo IP:".$$REMOTE_ADDR." alle ore: ".date("d-m-Y G:i:s")." \n";
17
18         @fputs($f,$frase);    // scrive la frase nel file tramite $f
19         @fclose($f);          // è importante chiudere il file
20
21         echo "..... Fatto!";
22
23     }else{
24         echo "<br>Errore durante l'apertura del file dati_6.txt";
25     }
26 ?>
27
28 </body></html>

```

Le modifiche sono semplici, è stato aggiunto un 'if' che controlla la dimensione del file 'dati_6.txt' se è maggiore del valore impostato nella variabile '\$max_file_size' apre il file in modalità 'w' e quindi ne cancella il contenuto, altrimenti lo apre in modalità 'a'. In entrambi i casi se il file non esiste viene creato.

5.6 Conclusioni

Il file system, come appena visto, permette di immagazzinare informazioni su disco dando la possibilità di aggiornarle modificandole o cancellandole. Tuttavia è bene tenere presente che esistono tecnologie relazionali apposite per la gestione di grandi moli di dati che garantiscono maggiori prestazioni e una migliore gestione dello spazio su disco. Tali tecnologie sono le basi di dati alle quali si accede tramite un linguaggio di interrogazione standard noto come SQL¹.

Il file rimane comunque uno strumento utile, semplice e immediato, e nel caso di piccole quantità di dati o nel caso in cui non si abbia accesso ad un DBMS², può essere preso in considerazione come alternativa alle basi di dati.

¹ Acronimo di "Structured Query Language" ovvero "linguaggio di interrogazione strutturato".

² Acronimo di "Data Base Management System " ovvero "Sistema di Gestione di Basi di Dati", è il software che si occupa di rispondere alle istruzioni SQL.

Classi e oggetti

Uno dei grandi vantaggi del PHP è la possibilità di scrivere codice orientato agli oggetti. Le classi sono particolari strutture che possono contenere al loro interno variabili e funzioni dette metodi. Una volta definita la classe è possibile definire uno o più oggetti del tipo classe, in questo modo gli oggetti, le variabili in essi contenute e i metodi saranno tutti indipendenti tra loro.

Ad esempio si immagina una classe *rubrica* che al suo interno contiene i metodi *aggiungi()*, *cerca()* e *rimuovi()*, per manipolare i dati in archivio basterà definire un oggetto di tipo rubrica, tutto il necessario sarà contenuto all'interno dell'oggetto, ovvero le procedure di inserimento, ricerca e cancellazione, lo sviluppatore dovrà solo fornire i parametri richiesti dai metodi.

Spesso è comodo usare all'interno di una classe metodi di un'altra classe, in PHP è possibile definire una classe come estensione di un'altra già esistente. Questo consente di risparmiare tempo e lavoro, inoltre rende il codice pulito e leggibile.

Si potrebbe avere la necessità di effettuare un confronto tra due voci in rubrica. Una soluzione è l'aggiunta di un metodo, un'altra è estendere la classe. La scelta non è sempre scontata, in questo caso la strada più semplice è la scrittura di un nuovo metodo *confronta()* ma spesso risulta conveniente creare una classe centrale che contiene i metodi fondamentali e comuni e varie classi satellite, estensioni di quella centrale, contenenti i metodi usati più di rado.

6.1 Definizione di classi e oggetti

Prima di approfondire il concetto di "oggetto" con degli esempi pratici è bene introdurre la sintassi PHP che permette di manipolarli.

Come più volte ripetuto un oggetto è un'istanza ad una classe, dunque la struttura della classe deve essere definita prima della dichiarazione dell'oggetto.

La sintassi di definizione di una classe è la seguente:

```
class nome_classe{

    var variabili_comuni_alla_classe ;

    function nome_classe(parametro1="valore_di_default1", ...){
        .
        .
        .
    }

    function nome_metodo1(parametro1, parametro2, ...){
        .
        .
        .
    }

    function nome_metodo2(parametro1, parametro2, ...){
        .
        .
        .
    }

}
```

L'istruzione **class** segnala l'inizio della dichiarazione della classe, segue il nome della classe e le parentesi graffe ('{ }') a delimitare l'inizio e la fine della struttura.

Come prima cosa vanno dichiarate le variabili locali, ovvero le variabili che saranno a disposizione di tutti i metodi e solo all'interno della struttura. Ovviamente, se non se ne ha bisogno possono essere omesse.

Di seguito va definito, se necessario, il costruttore, che è un metodo particolare il quale ha lo stesso nome della classe e viene eseguito al momento della creazione dell'oggetto. Il costruttore non è obbligatorio, una classe può anche non averlo, tuttavia nella maggior parte dei casi è comodo e si utilizza per eseguire delle operazioni "iniziali" necessarie al funzionamento dei metodi successivi.

Il costruttore può ricevere dei parametri che possono essere inizializzati a dei valori di default. Se al momento della dichiarazione dell'oggetto non vengono specificati valori diversi l'oggetto userà i valori stabiliti durante la dichiarazione del costruttore della classe.

Supponendo di avere una classe chiamata **'carrello'** il costruttore potrebbe essere qualcosa che inizializza le quantità a zero al momento della creazione dell'oggetto di tipo carrello:

```
class carrello{
    var $q, $t_spesa, $user;

    function carrello($quantita=0, $totale_spesa=0, $utente="anonimo"){
        $this->q = $quantita;
        $this->t_spesa = $totale_spesa;
        $this->user = $utente;
    }

    function aggiungi($quantita){
        $this->q += $quantita;
    }

    function svuota(){
        $this->q = 0;
        $this->t_spesa = 0;
        $this->user = "";
    }
}
```

Le variabili definite all'interno della classe saranno disponibili a tutti i metodi della stessa tramite l'uso del prefisso speciale **'\$this->'** che distingue le variabili e i metodi interni a "questa" classe.

Una volta definita la struttura della classe è possibile istanziare uno o più oggetti di quel tipo. La sintassi corretta per farlo è la seguente:

```
$spesa_brdp = new carrello;
$spesa_mirko = new carrello(0,1,"Mirko");
```

Sopra sono stati dichiarati due oggetti di tipo "carrello", nel primo caso i parametri di default sono stati lasciati invariati mentre nella seconda dichiarazione sono stati impostati parametri diversi da passare al costruttore per inizializzare il carrello.

Una volta creati, i due oggetti, sono entità separate e indipendenti, le variabili e i metodi dell'uno non possono in alcun modo influenzare quelli dell'altro. Questo è uno dei grandi vantaggi che si ha con la programmazione orientata agli oggetti oltre all'eleganza e alla semplicità del codice.

Non rimane che analizzare la sintassi per l'utilizzo dei metodi:

```
$nome_oggetto->nome_metodo(eventuali_parametri);
```

Come accennato la sintassi è intuitiva, nell'esempio sopra si potrebbe aggiungere una quantità al carrello virtuale tramite il metodo **'aggiungi(\$quantita)'**. Ecco come:

```
/*
    si suppone di ricevere una variabile in GET contenente
    la quantità di oggetti da inserire nel carrello. Il classico
```



```

    problema di un sito di commercio elettronico
*/

$q = $HTTP_GET_VARS["quantita"];

// aggiungo la quantità nel carrello
$spesa_brdp->aggiungi($q);

// la raddoppio e la inserisco anche nel secondo carrello
$q = $q * 2;      // oppure $q *= 2;
$spesa_mirko->aggiungi($q);

```

Supponendo che il visitatore voglia svuotare il proprio carrello cliccando su una voce particolare un parametro viene inviato ad una pagina PHP che richiama il metodo che elimina il contenuto del carrello. Ad esempio:

```

/*
    si suppone di ricevere una variabile in GET contenente
    il tipo di operazione da eseguire.
*/

if ($HTTP_GET_VARS["da_fare"]=="svuota"){

    // vuoto entrambi i carrelli
    $spesa_brdp->svuota();
    $spesa_mirko->svuota();
}

```

6.2 Utilizzo di classi e oggetti

In questo capitolo verrà trattato un esempio allo scopo di approfondire e consolidare la conoscenza della sintassi.

La classe seguente gestisce un minuscolo magazzino di automobili. Ovviamente non è di alcuna utilità applicativa ma aiuta ad acquisire dimestichezza con questo nuovo concetto.

```

0  <?
1  class automobile{
2
3      // Proprieta'
4      var $archivio;
5
6      // costruttore della classe
7      function automobile(){
8          $this->archivio = array("Audi"=>"A4", "Ford"=>"Focus, Fiesta", "Fiat"=>"Stilo",
9                                  "Lancia"=>"Libra", "Renault"=>"Laguna, Megane");
10     } // fine costruttore
11
12     function modello($marca){
13         // ritorna il modello della marca passata
14         return $this->archivio["$marca"];
15     } // fine metodo modello
16
17     function elenco(){
18         // visualizza l'elenco completo marca - Modelli
19         echo "<br>Elenco completo parco Auto:<br>";
20         foreach($this->archivio as $marca => $modello){
21             echo "<br>Per ".$marca." sono disponibili: ".$modello;
22         }
23     } // fine metodo elenco
24
25 } // Fine classe automobile
26
27 ?>
28

```

```

29 <html>
30 <body>
31
32 <?
33 // creo l'oggetto di tipo automobile!
34 $concessionario = new automobile;
35
36 if($HTTP_GET_VARS["casa"] != "0"){
37     echo "Ecco i modelli <u>".$HTTP_GET_VARS["casa"]."</u> disponibili:
38         <b>".$concessionario->modello($HTTP_GET_VARS["casa"])."</b>";
39 }else{
40     $concessionario->elenco();
41 }
42
43 ?>
44
45 <br><br><br>
46
47 <form method="get" action="<?=$PHP_SELF?>">
48     Seleziona la marca per avere i modelli disponibili:
49     <select name="casa">
50         <option value="0">Elenco completo
51         <option value="Audi">Audi
52         <option value="Ford">Ford
53         <option value="Fiat">Fiat
54         <option value="Lancia">Lancia
55         <option value="Renault">Renault
56     </select>
57
58     <input type="submit" value="Trova modelli!">
59 </form>
60
61 </body>
62 </html>

```

Dalla riga 1 alla 25 viene definita la nuova classe chiamata "automobile", che contiene una variabile `$archivio`, il costruttore `automobile` e due metodi `modello` ed `elenco`.

Come detto più volte, una classe al suo interno può contenere variabili e funzioni oltre a tutte le strutture messe a disposizione dal linguaggio.

Nell'esempio specifico il costruttore `automobile` non fa altro che riempire un array associativo assegnato alla variabile `$archivio`, tale variabile contiene i modelli delle varie marche disponibili nel magazzino.

Dopo il costruttore vengono definiti due semplici metodi: il primo, `modello($marca)`, fornisce i modelli della marca passata come parametro, leggendoli dall'array `$archivio` e restituendoli tramite il comando `return`, proprio come in una semplice funzione. Tutto questo viene fatto nella riga 14. Il secondo, `elenco`, non restituisce alcun valore e non richiede parametri al momento della chiamata ma visualizza a schermo l'intero magazzino tramite un ciclo `foreach` nelle righe 20, 21 e 22.

A questo punto la dichiarazione della classe è completa, non resta che istanziare uno o più oggetti di tipo "automobile" per poter sfruttare i metodi definiti. L'esempio continua proprio con l'utilizzo di un oggetto.

Alla riga 34 viene dichiarato l'oggetto `$concessionario` di tipo `automobile` tramite l'istruzione:

```
$concessionario = new automobile;
```

Al momento della dichiarazione viene eseguito il costruttore che, in questo caso, inizializza l'array `$archivio` (contenuto nel nuovo oggetto). Da questo momento sono disponibili tutti i metodi dell'oggetto `$concessionario`.

Nel caso specifico dell'esempio alle righe 38 e 40 vengono richiamati i due metodi definiti in precedenza. Tali metodi sono stati costruiti in modo diverso: il primo restituisce un valore tramite l'istruzione **'return'** mentre il secondo, **'elenco()'**, visualizza sullo schermo i dati senza ritornare alcun valore. Questa differenza nella definizione comporta un utilizzo diverso dei metodi.

Nel dettaglio, il metodo **'modello(\$marca)'** può essere associato ad una variabile o concatenato ad una stringa quindi può essere manipolato come una variabile. Il metodo **'elenco()'** invece, può essere usato solo in un modo, per visualizzare l'elenco di tutte le marche e modelli sullo schermo.

Ecco gli utilizzi dei metodi nel dettaglio:

```
$concessionario = new automobile;

// Concateno in una stringa
echo "Ecco i modelli <u>Audi</u> disponibili:
      <b>".$concessionario->modello("Audi")."</b>";

// Oppure assegno ad una variabile
$modelli = $concessionario->modello("Ford");

// Mentre il metodo elenco può essere usato solo così:
$concessionario->elenco();
```

Come detto gli oggetti sono entità separate, dunque è possibile istanziare più oggetti della stessa classe come:

```
$concessionario = new automobile;
$auto = new automobile;

// Concateno in una stringa
echo "Ecco i modelli <u>Audi</u> disponibili:
      <b>".$auto->modello("Audi")."</b>";

// Oppure assegno ad una variabile
$modelli = $concessionario->modello("Ford");

// Mentre il metodo elenco può essere usato solo così:
$concessionario->elenco();

// Mentre il metodo elenco può essere usato solo così:
$auto->elenco();
```

L'esempio continua con un semplice modulo HTML che spedisce una variabile contenente la marca selezionata a se stesso (tramite la variabile predefinita **'\$PHP_SELF'**). Tale variabile viene passata al metodo che restituisce i modelli disponibili per la marca scelta. Se il valore della marca è 0 viene visualizzato l'elenco completo tramite l'utilizzo del secondo metodo.

Il funzionamento dell'esempio può essere verificato a questo indirizzo: http://www.urcanet.it/brdp/php_manual/esempi/cap_7/1.php

6.3 Esempi applicativi

Una volta chiarita la sintassi si può passare a scrivere una prima classe da collezionare. Nella programmazione di tutti i giorni ci sono operazioni che si ripetono e che quindi possono risultare noiose. Un classico esempio è la gestione dei formati delle date. Il PHP mette a disposizione numerose funzioni di gestione delle date, esse sono versatili e accettano numerose opzioni. Potrebbe essere vantaggioso scrivere una classe che formatta la data secondo le proprie necessità, ovvero nei formati che si utilizzano più frequentemente. Ecco un esempio:

```

1  <?
2
3  // ----- DATA e ORA -----
4  /* Formatta la data in vari formati.
5   è possibile aggiungere nuovi formati accodando
6   righe alla "switch" finale
7  */
8
9  class data_ora{
10
11   // Proprieta'
12   var $mesi, $week;
13   var $ore, $minuti, $secondi;
14   var $gs, $mese, $gmese, $anno;
15
16   // costruttore
17   function data_ora(){
18       // Salva in diverse variabili la data nei vari formati:
19       // definisce l'array per i mesi e per i giorni della settimana.
20
21       $this->week = array(0=>"Domenica","Lunedì","Martedì","Mercoledì",
22                           "Giovedì","Venerdì","Sabato" );
23
24       $this->mesi = array ("1"=>"Gennaio", "Febbraio", "Marzo", "Aprile",
25                           "Maggio", "Giugno", "Luglio", "Agosto",
26                           "Settembre", "Ottobre", "Novembre", "Dicembre");
27
28       $this->gs = date("w"); // giorno della settimana 0 è domenica
29       $this->m = date("n"); // mese numerico 9 non 09
30       $this->mm = date("m"); // mese stringa 09 non 9
31       $this->g = date("j"); // giorno del mese numerico 9 non 09
32       $this->gg = date("d"); // giorno del mese stringa 09 non 9
33       $this->anno = date("Y"); // anno ovviamente numerico
34
35       $this->o = date("H"); // ore con lo zero avanti in formato 24
36       $this->mi = date("i"); // minuti con lo zero esempio: 09
37       $this->s = date("s"); // secondi con lo zero 09
38
39   } // fine costruttore
40
41
42   function formatta($tipo_data){
43
44       switch ($tipo_data) {
45           case 1:
46               return $this->g."-".$this->m."-".$this->anno;
47               break;
48           case 2:
49               return $this->anno."-".$this->m."-".$this->g;
50               break;
51           case 3:
52               return $this->m."-".$this->g."-".$this->anno;
53               break;
54           case 4:
55               return $this->week[$this->gs]." ".$this->g."-".$this->m."-
56                   ".$this->anno;
57               break;
58           case 5:
59               return $this->week[$this->gs]." ".$this->g."-".$this->mesi[$this->m]."-
60                   ".$this->anno;
61               break;
62           case 6:
63               return $this->week[$this->gs]." ".$this->gg."-".$this->mm."-
64                   ".$this->anno;
65               break;
66           case 7:
67               return $this->week[$this->gs]." ".$this->gg."-".$this->mesi[$this->m]."-

```

```

        ".$this->anno;
65     break;
66     case 8:
67         return $this->week[$this->gs]." ".$this->g."-".$this->mesi[$this->m]."-
            ".$this->anno."
68         ore: ".$this->o." ".$this->mi;
69     break;
70 }
71
72 } // fine metodo formattazione
73
74 }
75
76 ?>
77
78
79 <html>
80 <body>
81
82 <?
83
84     $adesso = new data_ora;
85
86     for ($i = 1; $i <= 8; $i++){
87         echo "<br>La data odierna formattata con <b>".$i."</b> è:
            ".$adesso->formatta($i);
88     }
89 ?>
90
91
92 </body>
93 </html>

```

Analizzando la classe nel dettaglio si vede che è composta da varie proprietà (le variabili), il costruttore ed un metodo. Il costruttore inizializza gli array contenenti i nomi dei giorni della settimana e dei mesi dell'anno, inoltre inizializza tutte le altre variabili con dei valori riferiti alla data "odierna" tramite la funzione predefinita `'date()'`. Tali variabili serviranno in seguito per estrarre i vari formati di data.

Il metodo `'formatta($tipo_data)'` in base al parametro passato restituisce una stringa contenente la data formattata in un particolare modo. Il parametro deve essere un intero nell'esempio compreso tra 1 e 8 ma nulla vieta di estendere l'elenco.

Il funzionamento è semplice, tramite una struttura `'switch'` (trattata nella sezione 2.5.4) il metodo concatena le variabili inizializzate dal costruttore in modo da generare una stringa particolare.

Per verificare il funzionamento della classe ed elencare tutti i formati delle date disponibili, l'esempio continua con la creazione di un oggetto `'$adesso'` di tipo `'data_ora'` e con un semplice ciclo `'for'` che richiama il metodo `'$adesso->formatta($i)'` passandogli i valori interi compresi tra 1 e 8 come vuole la dichiarazione della classe.

Il risultato dello script sarà simile a:

```

La data odierna formattata con 1 è: 19-12-2002
La data odierna formattata con 2 è: 2002-12-19
La data odierna formattata con 3 è: 12-19-2002
La data odierna formattata con 4 è: Giovedì 19-12-2002
La data odierna formattata con 5 è: Giovedì 19-Dicembre-2002
La data odierna formattata con 6 è: Giovedì 19-12-2002
La data odierna formattata con 7 è: Giovedì 19-Dicembre-2002
La data odierna formattata con 8 è: Giovedì 19-Dicembre-2002 ore: 10:40

```

Il metodo restituisce la data tramite l'istruzione `'return'`, quindi la chiamata al metodo può

essere trattata come una variabile, infatti, nell'esempio viene concatenata ad una stringa (riga 87) per essere visualizzata sullo schermo.

Al lettore il compito di comprendere il comportamento del metodo nel caso in cui venga fornito un parametro non intero oppure non compreso nell'intervallo di definizione della struttura di **'switch'**.

La prima parte di codice fino alla riga 77 potrebbe essere salvata in un file per diventare una sorta di libreria di seguito chiamata `'brdp_lib.php'` dove verranno raccolte tutte le classi di uso frequente. Una volta creato il file basterà includerlo all'inizio delle nuove pagine PHP per avere tutte le classi a disposizione. Dunque si potrà ottenere lo stesso risultato dell'esempio precedente creando il file `'brdp_lib.php'` e modificando l'esempio in questo modo:

```
<? include("brdp_lib.php");?>

<html>
<body>

<?
    $adesso = new data_ora;

    for ($i = 1; $i <= 8; $i++){
        echo "<br>La data odierna formattata con <b>". $i. "</b> è:
            ".$adesso->formatta($i);
    }
?>

</body>
</html>
```

In questo modo oltre ad essere più chiara la struttura è anche più semplice aggiornare le funzionalità delle classi. Se ad esempio la classe viene già usata in dieci pagine differenti e si ha la necessità di aggiungere un nuovo tipo di formattazione oppure modificarne uno già esistente, basterà modificare un unico file per rendere le modifiche disponibili in tutte e dieci le pagine che utilizzano oggetti di quel tipo. Al contrario si dovrebbe ripetere la stessa modifica in dieci pagine diverse.

Anche in questo caso l'esempio è raggiungibile all'indirizzo: http://www.urcanet.it/brdp/php_manual/esempi/cap_7/2.php

La classe appena vista potrebbe formattare anche date diverse da quella odierna, ad esempio ricevendo un parametro passato al costruttore e impostato come default alla data attuale, in questo modo, se specificato, al momento della dichiarazione dell'oggetto verrà formattata una data particolare, altrimenti verrà presa come data quella odierna. Questa semplice modifica viene lasciata al lettore come esercizio.

Nei capitoli seguenti verranno studiate altre classi ed inserite nella libreria `'brdp_lib.php'` appena creata.

6.4 Estensione di una classe

in fase di sviluppo...

Nelle prossime puntate

Questo è un capitolo che spero scomparirà presto. L'ho inserito solo per segnalarvi dove mi piacerebbe arrivare, ovvero tutto quello che mi piacerebbe trattare.

In questi anni di sviluppo di progetti in PHP ho raccolto oltre all'esperienza varie porzioni di codice che mi piacerebbe condividere con tutti quelli che, come me, ne hanno bisogno. Spesso ho perso molto tempo a risolvere problemi che con un piccolo suggerimento avrei superato in pochi minuti. Credo che con la condivisione del proprio lavoro si possa ottimizzare e migliorare il proprio codice fino a renderlo umanamente perfetto. Insomma credo nell'Open Source e spero che chi leggerà questi appunti e utilizzerà il mio codice, se non lo ha già fatto, si avvicini a questo modo di ragionare e di operare.

Dopo questo breve sfogo ecco gli argomenti che mi prefiggo di trattare:

- *Cookies e Session* ; per la gestione delle sessioni di navigazione;
- *Classi e Oggetti*; la programmazione ad oggetti è possibile e consigliata anche in PHP;
- *Upload file* ; è possibile anche tramite il protocollo HTTP
- *Interfacciamento con DBMS* ; come far comunicare il PHP con le più diffuse basi di dati;
- *Descrizione delle mie classi* ; condivisione e descrizione per l'utilizzo ed il perfezionamento delle mie classi;

Insomma questo è quello che mi piacerebbe fare... Il tempo non è molto e spero di riuscirci prima possibile.

Grazie a chi legge.

BRDP

Appendici

GNU GENERAL PUBLIC LICENSE

non modificabile

Testo originale: <<http://www.fsf.org/copyleft/gpl.html>>

GNU GENERAL PUBLIC LICENSE - Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE - TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means

either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may

not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODI-

FY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking

proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Indice analitico

- accesso sequenziale, 5.1
- append, 5.5
- array, 2.3, 2.3.5
- array(), 2.3.5
- boolean, 2.3, 2.3.2
- brdp_lib.php, 6.3
- break, 2.5.4, 2.5.9
- case, 2.5.4
- ciclo, 2.5.5, 2.5.6, 2.5.7, 2.5.8
- class, 2.3.6, 6.1, 6.2, 6.3
- classe, 6.1, 6.2, 6.3
- classi, 6
- classi estese, 6.4
- costruttore, 6, 6.1, 6.2, 6.3
- count(), 5.4
- date(), 1.1, 2.3.5, 5.4, 6.3
- DBMS, 5.6
- die(), 2.4.3
- do...while, 2.5.6
- double, 2.3.3
- echo, 1.1
- else, 2.5.2, 2.5.2
- elseif, 2.5.3, 2.5.4
- escape, 2.3.4
- estensione di una classe, 6.4
- extends, 6.4
- fclose(), 5.3, 5.4, 5.5
- feof(), 5.1, 5.3, 5.4
- fgets(), 5.3, 5.4
- file, 5
- file(), 2.4.3, 5.3
- filesize(), 5.5
- file_exists(), 5.3
- float, 2.3.3
- foalt, 2.3
- fopen(), 5.2, 5.3, 5.4, 5.5
- for, 2.5.7
- foreach, 2.5.8, 6.2
- form, 3
- fputs(), 5.4, 5.5
- function, 6.1, 6.2, 6.3
- function(), 4.2, 4.3
- funzioni, 4, 4.1
- get, 3
- get, 3.1
- gettype(), 2.1, 2.3
- global, 4.2
- hash, 2.3, 2.3.5
- if, 2.3.2, 2.5.1
- include(), 2.5, 2.5.10, 4

- inclusione di file, 2.5.10
- integer, 2.3, 2.3.1
- intero, 2.3.1
- interruzione, 2.5.9
- is_array(), 2.5.2
- is_double(), 2.5.2
- is_integer(), 2.5.2
- is_string(), 2.5.2
- libreria, 6.3
- loop, 2.5.5, 2.5.6, 2.5.7, 2.5.8
- metodi, 6
- metodo, 6.1, 6.2, 6.3
- new, 6.1, 6.2, 6.3
- numero a virgola mobile, 2.3.3
- object, 2.3.6
- oggetti, 6
- oggetto, 6.1, 6.2, 6.3
- operatori, 2.4
 - operatori aritmetici, 2.4.1
 - operatori di assegnazione, 2.4.2
 - operatori di confronto, 2.4.6
 - operatori di controllo degli errori, 2.4.3
 - operatori di incremento e decremento, 2.4.4
 - operatori logici, 2.4.5
- phpinfo(), 1.1, 2.2
- post, 3.2
- post, 3
- post_max_size, 3.3
- print_r(), 2.3.5
- return, 4.2, 4.3, 6.2, 6.3
- ricorsione, 4.3
- settype(), 2.1, 2.3
- SQL, 5.6
- string, 2.3, 2.3.4
- stringa, 2.3.4
- Strutture di controllo, 2.5
- substr(), 4.1
- switch, 2.5.4
- unlink(), 5.4
- var, 6.1, 6.2, 6.3
- variabili globali, 4.2
- Variabili http, 2.2
- variabili locali, 4.2
- while, 2.5.5
- \$HTTP_GET_VARS, 3.4
- \$HTTP_POST_VARS, 3.4
- \$HTTP_REFERER, 2.2
- \$HTTP_USER_AGENT, 2.2
- \$PHP_SELF, 6.2
- \$REMOTE_ADDR, 2.2, 5.4
- \$SCRIPT_NAME, 2.2
- \$SERVER_NAME, 2.2

\$this->, 6.1, 6.2, 6.3