

HACKER ITALIA EDITIONS SRL



PHP & MYSQL

GUIDA ALLA
PROGRAMMAZIONE
PHP E MYSQL

YOUNES HAOUFADI

Manuale PHP e

MYSQL da 0

Younes Haoufadi

Scritto da: Younes Haoufadi, il 31 gennaio 2018

Editore: Hacker Italia Srl

Copyrighted Material By Hacker Italia Srl

Contacts:

Hacker Italia Srl a YITW Company in Milan:

Hacker Italia Srl, Via Alcide De' Gasperi 92, Rho, MI
20017. Services@hacker-italia.com +39 02 87 15 9225

Hacker-italia.com

YITW Incorporated Italy: Via Gaetano de Castilia, 1,
20124 Milano MI italy@yitw.com +39 02 87 13 4543

Yitw.com

YITW Inc.:

77 King St W, Toronto, ON M5K 1A2, Canada

info@yitw.com +1 416-869-1079 Yitw.com

Questo libro è distribuito da Amazon Media, Amazon Inc. e Affiliate, Streetlib Srl e distributori, unici distributori ufficiali. Ogni distribuzione non ufficiale verrà segnalata come violazione di Copyright.

© 2018 Hacker Italia Srl

Introduzione:

Lo scopo è quello di consentire a chi non conosce questo magnifico di linguaggio di impararlo facilmente e gratuitamente tramite questa semplice guida,

Seguite attentamente tutti i

tutorial: sono ordinati,
numerati e vanno seguiti passo
dopo passo (non saltate nessun
capitolo se veramente volete
imparare il linguaggio!)

Il php è stato sviluppato nel
1994, è un linguaggio lato
server, è simile al C e al Perl

ed è il maggior linguaggio di sviluppo del web.

E' importante ricordare la sua interazione con database come MYSQL.

Ambienti di lavoro :

Prima di iniziare a seguire questa guida e' strettamente

consigliata una conoscenza minima di html/ftp. Possedere dunque:

Un qualsiasi editor html(Es. Dreamwavare, Golive, anche il blocco note va bene) ;

Uno spazio web con connessione ftp e supporto

php(es. lo puoi avere gratis
tramite altervista.org), oppure
in alternativa puoi installare
server locali come Xamp
Server.

Importante notare come
L'HTML sia integrabile
facilmente nel php.

Le pagine php hanno
estensione .php.

Questa guida ha la scopo di
insegnare a pieno le basi e
l'essenziale del php. Buona
Fortuna, proseguite al Primo
Capitolo.

BUONA LETTURA!

Cos'è PHP?

PHP è un potentissimo linguaggio di scripting che consente di creare complesse applicazioni lato server (che girano cioè all'interno di un web server) come ad esempio forum, guestbook, sistemi di statistiche, e-commerce, ecc.

Da un punto di vista tecnico possiamo dire che un server web è in grado di "far girare" applicazioni in PHP solo nel momento in cui sia stato installato il relativo interprete il quale ha il compito di leggere la sintassi PHP e trasformarla in linguaggio macchina.

Prima di procedere è bene sottolineare

che PHP è un linguaggio multiplatforma, questo significa che funziona correttamente su server web equipaggiati con differenti sistemi operativi in quanto esistono differenti versioni dell'interprete PHP in grado di funzionare sia in ambiente UNIX che Win. In linea di massima possiamo dire che è consigliabile un suo utilizzo su server [Linux](#) (che costituisce l'ambiente nativo di questo linguaggio) tuttavia, come detto, è possibile utilizzarlo anche in ambiente Windows (se adeguatamente equipaggiato).

Creare l'ambiente di lavoro

Se avete [Windows](#) e volete testare in locale gli esercizi pratici che vi

proporrò in questo corso vi consiglio di scaricare [WAMP](#) oppure [EasyPHP](#): si tratta di due pacchetti grazie ai quali potrete installare in automatico sul vostro PC Windows il WebServer Apache, PHP e MySQL.

Se utilizzate MAC OS X potete fare altrettanto scaricando [MAMP](#), mentre se utilizzate una piattaforma Linux è probabile che tutto ciò che vi serve sia già installato nel sistema. In alternativa vi consiglio di visitare i seguenti siti dove potrete scaricare tutto quello di cui avete bisogno:

- [Apache](#)

Il sito ufficiale del WebServer più diffuso al mondo. Freeware;

- [PHP](#)

Il sito ufficiale di Php dove potrai scaricare gratis l'ultima release di PHP. Freeware;

- [MySQL](#)

Il sito ufficiale del database preferito dagli sviluppatori PHP. Potente e Gratuito.

La procedura per l'installazione di questi software varia a seconda della versione di Linux che state utilizzando. Vi consiglio di seguire attentamente le informazioni contenute nei manuali dei rispettivi software.

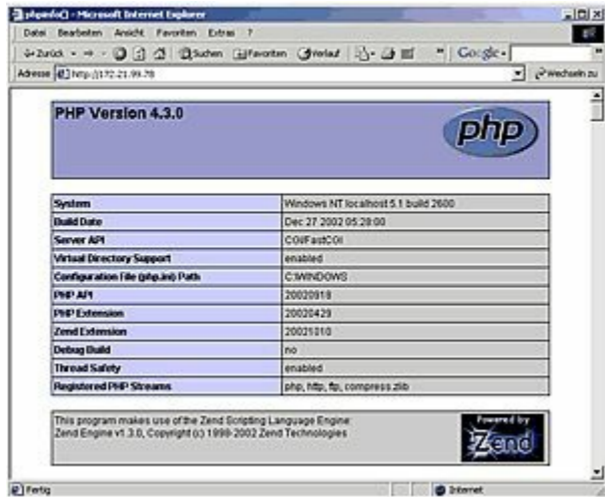
Una volta terminata la procedura di installazione possiamo testarne il buon esito con poche righe di PHP. Aprite un comune editor di testo (NotePad di Windows va benissimo!) e scrivete:

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

Salvate come "info.php" nella root del webserver. Ora digitate:>/p>

<http://localhost/info.php>

e state a guardare cosa succede... se tutto è andato liscio dovrete vedere una schermata simile a questa:



The screenshot shows a Microsoft Internet Explorer browser window with the address bar set to `http://172.21.99.78`. The page content includes the PHP logo and a table of system information.

PHP Version 4.3.0	
System	Windows NT localhost 5.1 build 2600
Build Date	Dec 27 2002 05:28:00
Server API	CGIFastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
PHP API	20020918
PHP Extension	20020429
Zend Extension	20021010
Debug Build	no
Thread Safety	enabled
Registered PHP Streams	php, http, ftp, compress.zlib

At the bottom of the page, there is a text box stating: "This program makes use of the Zend Scripting Language Engine: Zend Engine v1.3.0, Copyright (c) 1998-2002 Zend Technologies" and a logo for "Powered by Zend".

Ora che abbiamo installato PHP possiamo iniziare a vedere un po' come

funziona...

A partire da questo capitolo vedremo come scrivere le prime linee di codice nel **linguaggio di PHP**, vedremo, in pratica, come definire delle istruzioni che dovranno essere poi tradotte dal web server (attraverso l'interprete in esso installato): si parla, in questo caso, di *scripting server-side* in quanto l'elaborazione della pagina avviene - appunto - a livello di server, mentre il client riceve un semplice output HTML "preformattato".

Proprio per questa sua capacità di creare "al volo" l'output HTML, il PHP è definito **linguaggio dinamico**, esattamente come ASP, JSP, ecc.

Questa sua capacità di produrre un output "variabile", infatti, rende il PHP molto diverso dal semplice HTML, non avendo quest'ultimo la capacità di dinamizzare il suo output (se non ricorrendo ad altri linguaggi di scripting).

I tag di rendering di PHP

Ma veniamo al codice! Apriamo il nostro blocco note (o un qualsiasi altro editor di testo o specifico per PHP) e scriviamo la nostra prima riga di codice

PHP:

```
<?php
```

```
echo "Ciao... questo è PHP!";
```

```
?>
```

Saviamo tutto come "prova.php" e testiamo il risultato all'interno del vostro web server.

Come avrete capito ogni porzione di codice PHP si apre con **<?php** e si chiude con **?>** (cosiddetti **tag di rendering**) che indicano al nostro interprete PHP, rispettivamente, l'inizio del codice e la sua fine. Tutto ciò che è compreso tra **<?** e **?>** è PHP, tutto ciò che sta fuori verrà restituito dal server come semplice testo o HTML.

Questa osservazione, che può sembrare banale, è in realtà molto importante: all'interno dello stesso file, infatti, è possibile alternare blocchi di codice PHP con porzioni di semplice HTML, come nell'esempio qui sotto:

```
<p>Questo è puro HTML...</p>  
  
<?php  
  
echo "<p>Mentre questo è PHP!</p>";  
  
?>
```

Ovviamente, perchè tutto ciò funzioni, è necessario che il nostro file abbia estensione **.php**, in caso contrario, se utilizzassimo l'estensione *.html*, il codice PHP non verrebbe "tradotto" dal web-server ma semplicemente mostrato

a video!

Nel nostro esempio i tag di rendering sono stati scritti sopra e sotto il codice PHP, ma nulla vieta di scrivere tutto in linea in questo modo:

```
<?php echo "Ciao... questo è PHP!"; ?>
```

Una simile sintassi, seppur corretta, è - ad avviso di chi scrive - da utilizzarsi solo in situazioni particolari (ad esempio quando, come nel nostro esempio, si deve utilizzare un'unica istruzione) essendo preferibile aggiungere un ritorno a capo dopo l'apertura e subito prima della chiusura dei tag di rendering al fine di agevolare la leggibilità del codice.

Stampare a video: echo

e print

Nello specifico dell'esempio che abbiamo visto sopra, abbiamo scritto una sola linea di codice: abbiamo utilizzato la funzione **echo** per dire al nostro server di "stampare a video" la frase contenuta tra le virgolette:

```
Ciao... questo è PHP!
```

Alla fine della nostra istruzione abbiamo poi usato il punto e virgola (;) che serve a chiudere la riga (o, più propriamente, a terminare un'istruzione PHP).

E' il caso di sottolineare che, al posto di **echo**, avremmo potuto usare **print** per assolvere alla stessa funzione:

```
<?php
```



```
print "Ciao... questo è PHP!";
```

```
?>
```

I due commadi, infatti, salvo differenze marginali, servono al medesimo scopo e possono essere usati indifferentemente per stampare qualcosa a video.

Il nostro esempio, ovviamente, è puramente scolastico: l'utilizzo di PHP, infatti, è fine a se stesso e non ha alcuna utilità a livello applicativo. Nelle prossime lezioni, ovviamente, vedremo come combinare il comando echo/print in situazioni più complesse ed articolate dove l'output da stampare a video viene prodotto attraverso l'utilizzo di variabili, condizioni o altri tipi di elaborazione.

Inserire commenti al codice PHP

Prima di proseguire e di addentrarci in aspetti più articolati della **programmazione con PHP**, vorrei soffermarmi sull'importanza di inserire, con costanza e continuità, **commenti al codice** all'interno degli *script* che andremo a realizzare con questo linguaggio.

I commenti al codice sono delle "note" che il programmatore aggiunge, per

comodità, al codice stesso. Si tratta di "appunti", semplice testo che non viene né elaborato né stampato a video: in pratica si tratta di riferimenti interni che possono essere letti solo dallo sviluppatore che lavora sul codice sorgente dello script.

Ad esempio, potrebbe essere utile scrivere sopra ad una determinata funzione qual'è il suo compito o come deve essere utilizzata nel contesto applicativo, oppure il perchè di una determinata sequenza di passaggi necessari per raggiungere un certo risultato...

I commenti sono fondamentali (e non devono essere trascurati) per una serie di motivi:

- perché semplificano interventi al codice effettuati in un momento successivo alla sua creazione (dopo molto tempo, infatti, certi passaggi logici potrebbero apparire meno chiari);
- perché facilitano il compito di altre persone chiamate a lavorare sul nostro codice.

Come scrivere un commento in PHP

Scrivere un commento all'interno del codice PHP è molto semplice... basta iniziare una riga con `//` oppure con `#`. Tutto quello che ne viene posizionato a destra non verrà processato da PHP e pertanto non avrà alcun effetto durante

l'elaborazione. Facciamo un esempio:

```
<?php
```

```
//Uso echo...
```

```
echo "ciao...";
```

```
#Ed ora uso print!
```

```
print "ciao...";
```

```
?>
```

Oppure:

```
<?php
```

```
echo "ciao..."; //Uso echo...
```

```
print "ciao..."; #Ed ora uso print!
```

?>

Quelli visti sino ad ora sono commenti su singola linea, qualora, invece, volessimo scrivere commenti su più righe potremo usare questa sintassi:

```
/*
```

```
questo è un commento su più righe...
```

```
questo è un commento su più righe...
```

```
questo è un commento su più righe...
```

```
*/
```

```
echo "ciao...";
```

Come avrete notato è considerato commento tutto ciò che è compreso tra /* e */ a prescindere che occupi una

o più linee.

Le variabili in PHP

Uno degli elementi base di ogni linguaggio di programmazione

(compreso PHP) sono certamente **le variabili**. La variabile può essere definita come un'area di memoria in cui vengono salvate delle informazioni (a cui il programmatore assegna un particolare identificatore) che possono mutare durante la fase di elaborazione del programma stesso.

In PHP tutte le variabili iniziano con il simbolo dollaro (\$). Il valore di una variabile viene assegnato con il simbolo uguale (=).

Vediamo un esempio di variabile testuale (variabile di tipo stringa):

```
// posso usare le virgolette
```

```
$variabile = "contenuto...";
```



```
// oppure l'apice
```

```
$variabile = 'contenuto...';
```

Qualora la vostra variabile avesse valore numerico non sarebbero necessarie le virgolette; ecco un esempio:

```
// numero intero
```

```
$variabile = 7;
```

```
// numero decimale
```

```
$variabile = 7.3;
```

Vediamo adesso un semplice codice PHP che sfrutti le variabili ed il comando echo per stampare a video,

appunto, il valore della variabile:

```
<?php
```

```
$variabile = "Testo da stampare";
```

```
echo $variabile;?>
```

Attenzione: i nomi di variabile sono case sensitive... attenzione quindi a maiuscole e minuscole!

Facciamo un altro esempio:

```
<?php
```

```
$oggetto = "casa";
```

```
$colore = "rosso";
```

```
echo "La " . $oggetto . " è di colore " .  
$colore;
```

?>

Nell'esempio qui sopra abbiamo utilizzato due variabili e le abbiamo stampate a video all'interno di una frase. Come avrete notato abbiamo usato il punto (.) per unire tra loro le diverse parti della frase. Il punto, infatti, è usato in PHP come elemento di concatenazione. Bisogna ricordare in questa sede che il PHP prevede anche un altro modo di concatenare le stringe all'interno del comando echo() che è questo:

```
<?php
```

```
$oggetto = "casa";
```

```
$colore = "rosso";
```

echo "La \$oggetto è di colore \$colore";

?>

Quest'ultimo metodo, seppur più semplice, può tuttavia presentare dei problemi in situazioni particolari che non ritengo opportuno trattare in questa sede. Il mio consiglio, quindi, è di utilizzare la forma vista sopra (è più corretta) che fa uso del punto per concatenare le diverse parti della stringa da restituire in output.

Le costanti in PHP

Una *costante*, come lascia intuire il nome, è una porzione di memoria il cui contenuto non cambia durante la fase di elaborazione del nostro programma in PHP. A differenza della *variabile*, che una volta definita può cambiare valore, la *costante* resta sempre uguale a se stessa ed ogni "tentativo" di cambiarne il valore produrrà un errore.

Per definire una costante in PHP si utilizza la funzione **define()** in questo modo:

```
define('NOME_COSTANTE','valore della costante');
```

Valgono per le costanti le stesse osservazioni fatte per le variabili:

- Se vengono assegnati valori di tipo stringa questi dovranno essere racchiusi in apici o doppi apici;
- Se vengono assegnati valori numerici non servono gli apici.

Qualche esempio:

```
define('NOME_COSTANTE_N1','valore  
della costante');
```

```
define("NOME_COSTANTE_N2","valo  
della costante");
```

```
define('NOME_COSTANTE_N3',123);
```

```
define('NOME_COSTANTE_N4',123.45
```

Per consuetudine, solitamente, i nomi delle costanti vengono scritti in maiuscolo (ma nulla vieta di usare il minuscolo). Si noti che i nomi delle

costanti, esattamente come accade nelle variabili, sono *case-sensitive*, quindi si faccia attenzione all'utilizzo di maiuscole e minuscole nel nome della costante.

E' anche possibile definire costanti *case-insensitive*, in tal caso sarà sufficiente specificare 'true' come terzo parametro nella funzione *define()*:

```
// definisco una costante case insensitive
```

```
define('NOME_COSTANTE','valore  
della costante',true);
```

```
echo NOME_COSTANTE; // corretto
```

```
echo nome_costante; // corretto
```

```
echo Nome_Costante; // corretto
```

Come abbiamo visto nell'esempio qui sopra la costante viene utilizzata nel codice PHP semplicemente mediante il proprio nome **senza, cioè, il simbolo del dollaro** che è tipico delle variabili.

Con PHP possiamo gestire facilmente anche le operazioni matematiche. Vediamo insieme gli **operatori matematici** da utilizzare:

Operatore	Operazione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione

L'unico operatore matematico a richiedere qualche precisazione è, probabilmente, l'operatore Modulo il quale serve per calcolare il resto di una divisione. Ad esempio:

$$5 \% 4 = 1$$

(il 4 sta nel 5 una sola volta con il resto, appunto, di 1).

Facciamo qualche esempio di calcoli con PHP usando anche le variabili.

Nell'esempio che seguirà faremo un semplice calcolo della spesa: poniamo di comprare 4 mele a 50 centesimi di Euro l'una... quanto abbiamo speso? Vediamolo con un semplice script PHP:

```
<?php
```

```
$mele = 4;
```

```
$euro = 0.5;
```

```
$totale = $mele*$euro;
```

```
echo "Abbiamo speso " . $totale . "  
Euro";
```

```
?>
```

Nota: nell'esempio abbiamo usato due variabili numeriche: una (`$mele`) di tipo `integer` (numero intero) e l'altra (`$euro`) di tipo `double` (numero decimale).

PHP consente di incrementare con facilità il valore di una variabile. Se ad esempio volessimo incrementare il valore della variabile \$totale dell'esempio visto sopra faremmo così:

```
$totale = $totale + 5;
```

il che equivale a: `$totale += 5;`

Dopo tale operazione la nostra variabile \$totale non avrà più il valore originario, bensì quello nuovo dato dalla maggiorazione di 5.

Ancora, PHP prevede delle scorciatoie per incrementare o decrementare di una unità una data variabile numerica; ma torniamo al solito esempio di prima... Per incrementare di uno (+1) il valore

della variable \$totale possiamo scrivere semplicemente:

```
$totale++;
```

Per decrementare il valore di uno (-1) useremo:

```
$totale--;
```

Per finire il nostro breve escursus ricordatevi che nel gestire le operazioni matematiche con PHP è importante... rispettare le regole matematiche! Facciamo un esempio:

```
$calcola = 5+7*2;
```

è diverso da:

```
$calcola = (5+7)*2;
```

Ricordatevi quindi di rispettare le "le priorità" del calcolo matematico

mediante un uso corretto delle parentesi!

Facciamo il punto della situazione: fin ora abbiamo visto cos'è il PHP, come implementare le prime righe di codice, come creare e gestire variabili e costanti (di tipo stringa o numerico) e, per finire come compiere operazioni matematiche con le variabili numeriche.

In questo capitolo, prettamente pratica, vedremo come è possibile combinare il codice PHP con il comune HTML. Il PHP, infatti, è un **liguaggio HTML-embedded**, consente cioè di unire all'interno dello stesso file sia istruzioni di PHP che semplice codice HTML.

Facciamo un esempio di pagina .php "mista" composta cioè sia di PHP che di HTML.

```
<html>
<head>
<title>PHP e HTML</title>
</head>
<body>
```

Questo è `HTML...`

```
<?php
```

```
echo "Mentre questo è PHP!";
```

```
?>
```

```
</body>
```

```
</html>
```

Ovviamente la pagina in questione andrà salvata con estensione `.php` (e non `.html`).

Facciamo un'altro esempio che sfrutti anche le variabili. Per comodità riprendiamo l'esempio visto nel capitolo precedente.

```
<?php  
$mele = 4;  
$euro = 0.5;  
$totale = $mele*$euro;  
?>  
  
<html>
```



```
<head>
```

```
<title>PHP e HTML</title>
```

```
</head>
```

```
<body>
```

La mamma ha comprato `<?php echo $mele; ?>` al costo di `<?php echo $euro; ?>` l'una.`
`

In totale ha speso `<?php echo $totale; ?>` Euro!

```
</body>
```

```
</html>
```

Come vedete, dopo aver definito il valore delle variabili all'interno di una

porzione di codice PHP (che abbiamo posizionato all'inizio del documento), le abbiamo "stampate" all'interno del documento HTML usando una semplice sintassi:

```
<?php echo $variabile; ?>
```

all'interno del comune codice HTML.

Gli operatori di confronto

Gli operatori di confronto consentono, appunto, di effettuare dei confronti tra valori al fine di prendere determinate "decisioni" durante l'esecuzione del codice. Attraverso questi operatori, infatti, si realizzano le cosiddette **strutture di controllo** cioè le

strutture logiche che consentono ad un'applicazione di fare una cosa oppure un'altra in base al verificarsi o meno di una data condizione.

Quando utilizziamo questi operatori confrontiamo i due valori posti a sinistra e a destra dell'operatore stesso. Il risultato di questa operazione sarà, ogni volta, vero (*true*) o falso (*false*). Questi operatori sono:

Operatore	Operazione
=	Uguale
!=	Diverso
<	Minore
>	Maggiore

<=	Minore o Uguale
>=	Maggiore o Uguale

Facciamo qualche esempio:

```
//definiamo due variabili
```

```
$a = 7;
```

```
$b = 5;
```

```
//facciamo qualche confronto...
```

```
$a == $b; //Falso
```

```
$a != $b; //Vero
```

```
$a < $b; //Falso
```

Gli operatori di confronto non operano soltanto con le variabili numeriche, ma anche con le variabili di tipo stringa.

In questo caso il confronto viene effettuato sull'ordine alfabetico dei caratteri: vale a dire che vengono considerati "minori" i caratteri che "vengono prima" nell'ordine alfabetico (quindi la lettera "a" è minore di "b", "b" è minore di "c" e così via). Inoltre tutte le lettere minuscole sono sempre considerate "maggiori" delle lettere maiuscole ("a" è maggiore di "A"). Per finire bisogna ricordare che tutte le lettere sono considerate "maggiori" dei numeri compresi tra 0 e 9 (quindi "a" è maggiore di 7).

In realtà confronti di questo tipo hanno un significato abbastanza relativo, in quanto difficilmente vi capiterà di utilizzarli.

Un caso particolarissimo è quello in cui vengono confrontate una variabile tipo stringa ed una numerica. In quest'ultimo caso PHP cercherà di trasformare la variabile stringa in numerica: per fare questo PHP controlla se all'inizio della stringa ci sono dei numeri, se ne trova, considererà tutti i numeri che trova inizialmente come il valore numerico di quella stringa; se non ne trova, il valore della stringa sarà uguale a zero. Facciamo un esempio:

```
//definiamo due variabili
```

```
$a = 2;
```

```
$b = 2 amici;
```

```
//facciamo un confronto...
```

```
$a == $b; //Vero
```

Gli operatori logici

Gli operatori logici servono letteralmente per combinare, alternare o negare più valori booleani (true/false).

Operatore	Operazione
AND	Tutte gli operatori dev veri. In alternativa a " può usare "&&"

OR

Almeno uno dei due c
deve essere vero. In a
"OR" si può usare "||"

XOR

Solo uno dei due oper
essere vero (l'altro de
falso)

NOT

E' l'operatore di nega
usa con un solo opera
sostanza effettua una
restituisce vero quand
l'operatore è falso, e v

Ecco qualche esempio:

```
(10 > 4) And (2 == 2); //vero: entrambe le  
condizioni sono vere
```


`(4 < 9) Or (8 > 12); //Vero: la prima condizione è vera`

`NOT (4 < 9); //Falso: la negazione ribalta il valore del paragone` L'importanza degli operatori di confronto e logici diventerà evidente nella prossimo capitolo quando parleremo delle strutture di controllo di PHP.

Le condizioni: if, else e switch

Questo capitolo del nostro corso su PHP è davvero fondamentale! Vedremo infatti come gestire il comportamento del nostro script sulla base del verificarsi o meno di determinate condizioni: qualora

una data condizione si verifichi (sia vera) il codice si comporterà in un modo, nel caso contrario (falsa) si comporterà in un altro.

Tutto ciò è realizzato mediante quelle che, nel titolo di questo capitolo, abbiamo chiamato "condizioni" anche se, in realtà, è decisamente più corretto parlare di **strutture di controllo o strutture condizionali**.

If...else

La principale di queste strutture di controllo è il costrutto **if..else** (in italiano "se"..."altrimenti"). La struttura logica del suo funzionamento è davvero banale e ricalca totalmente la logica del comune utilizzo grammaticale della

particella ipotetica "se".

Facciamo un esperimento: traduciamo in PHP la frase "se piove resto a casa":

```
<?php  
  
//specifico il valore della variabile  
$tempo  
  
$tempo = "piove";  
  
  
//Costruisco la condizione  
  
if ($tempo == "piove"){  
    echo "resto a casa";  
}  
  
?>
```

Nel codice qui sopra la condizione si realizza e quindi "resto a casa" :-)

Si noti che questa semplicissima condizione avrebbe potuto essere scritta anche su una sola linea, pertanto senza utilizzare le parentesi graffe:

```
if ($tempo == "piove") echo "resto a casa";
```

Le parentesi graffe, infatti, sono utilizzate per racchiudere una pluralità di istruzioni all'interno del medesimo blocco logico.

Nota: se nella vostra tastiera non figurano le **parentesi graffe** usate i tasti sui quali sono raffigurate le parentesi quadre, tenendo premuto ALT GR + il tasto per le maiuscole.

Ora facciamo un altro passo ancora e traduciamo in codice questa frase: "se piove resto a casa, altrimenti vado al parco":

```
<?php
//specifico il valore della variabile
$tempo
$tempo = "sole";

//Costruisco la condizione
if ($tempo == "piove"){
    echo "resto a casa";
} else {
```

```
echo "vado al parco";  
}  
?>
```

Nota: ricordatevi la differenza tra "=" e "==". Il primo è un operatore di assegnazione, il secondo è un operatore di confronto. Non confondeteli altrimenti non funzionerà nulla!

Anche questa volta poteva essere scritto tutto senza parentesi graffe:

```
if ($tempo == "piove") echo "resto a casa";
```

```
else echo "vado al parco";
```

Vediamo adesso qualcosa di ancora più complesso: "se piove e fa freddo resto a casa, se non fa freddo vado al bar. Se invece non piove vado al parco":

```
<?php
```

```
//specifico il valore della variabile
```

```
$tempo
```

```
$tempo = "sole";
```

```
//...e quello della variabile $temperatura
```

```
$temperatura = "freddo";
```

```
//Costruisco la condizione
```

```
if ($tempo == "piove"){  
    if ($temperatura == "freddo"){  
        echo "resto a casa";  
    }else{  
        echo "vado al bar";  
    }  
}else{  
    echo "vado al parco";  
}  
?>
```

Quello che abbiamo fatto è stato semplicemente costruire una **struttura**

idificata di condizioni.

Nel codice qui sopra abbiamo creato una sorta di duplice livello di condizione (il secondo è spostato verso destra). Se la prima condizione si verifica se ne innesta una seconda. Qualora invece la prima condizione non dovesse verificarsi la seconda condizione non viene nemmeno verificata.

Switch

Gli stessi risultati del costrutto "if..else" si possono ottenere con il controllo **switch**.

Per capire meglio il funzionamento di switch passiamo subito ad un esempio pratico e traduciamo in codice questa

frase: "se piove resto a casa, se c'è il sole vado al parco, in ogni altro caso vado al bar". Ecco il codice:

```
<?php
//specifico il valore della variabile
$tempo

$tempo = "sole";

switch ($tempo) {

    //se piove...

    case 'piove':

        echo "resto a casa";

        break;
```

```
//se c'è il sole...
```

```
case 'sole':
```

```
    echo "vado al parco";
```

```
    break;
```

```
//negli altri casi...
```

```
default:
```

```
    echo "vado al bar";
```

```
}
```

```
?>
```

L'istruzione "switch" prevede che venga indicata fra parentesi un'espressione che servirà come elemento discriminante

(nel nostro caso la variabile \$tempo); di seguito, tra parentesi graffe, scriviamo le diverse possibilità offerte dall'espressione iniziale: nel momento in cui ne viene trovata una "vera", PHP esegue il codice indicato di seguito, fino a quando non incontra un'istruzione "break".

Per finire (sempre tra le parentesi graffe) è possibile indicare il "default" che indica a PHP il comportamento da tenersi qualora nessuna delle condizioni indicate sopra si realizzi. L'indicazione "default" può anche essere assente, ma quando c'è deve essere l'ultima della switch.

I cicli: for, while e do while

Chiunque abbia già delle semplici nozioni di programmazione con altri linguaggi di scripting saprà certamente di cosa si tratta e quale grande importanza rivestono i **cicli** (o *iterazioni*). Tuttavia, trattandosi di una guida di base, ritengo opportuno introdurre l'argomento partendo da una semplice definizione che chiarisca le idee anche a chi è a digiuno dei concetti più basilari.

Ciclo for

Un ciclo, come già suggerisce la parola,

consiste semplicemente in una **ripetizione di un dato comando per un dato numero di volte**.

Come al solito partiamo da un esempio: facciamo finta di essere tornati ai banchi della scuola elementare e di studiare le tabelline. Il compito per domani è scrivere la tabellina del numero 3. Bene... vediamo come farlo con l'aiuto di PHP e dei cicli appunto:

```
<?php
for ($moltiplicatore = 1; $moltiplicatore
<= 10; $moltiplicatore++) {
    $risultato = 3 * $moltiplicatore;
    echo "3 * " . $moltiplicatore . " = " .
$risultato . "<br/>";
```

```
}
```

```
?>
```

Nell'esempio qui sopra abbiamo utilizzato il **ciclo for** accompagnato (fra parentesi) dalle istruzioni che delimitano il ciclo; di seguito, si racchiudono fra parentesi graffe tutte le istruzioni oggetto di ripetizione.

Come abbiamo detto, all'interno delle parentesi tonde abbiamo inserito le istruzioni: queste sono tre e sono divise tra loro da un punto e virgola.

La prima istruzione (`$moltiplicatore = 1`) viene eseguita una sola volta all'inizio del ciclo.

La seconda (`$moltiplicatore <= 10`) è la condizione che viene valutata prima di

ogni nuova iterazione del ciclo. Il ciclo prosegue fino a quando questa condizione è vera. Quando risulterà falsa il ciclo termina.

La terza (`$moltiplicatore++`) viene eseguita al termine di ogni iterazione del ciclo e serve, appunto, per proseguire nello stesso finchè la condizione lo consente.

Nota: lavorando con i cicli, è molto importante stare attenti a non creare mai una situazione di "loop" la quale consiste nell'infinito re-iterarsi di un ciclo che non incontra mai la fine. Tra le altre cose ciò può causare dei seri problemi anche allo stato di salute del nostro server!

while

Dopo aver visto come funziona il ciclo for passiamo a vedere come raggiungere gli stessi risultati utilizzando **while**.

```
<?php  
  
$moltiplicatore = 1;  
  
while ($moltiplicatore <= 10) {  
  
    $risultato = 5 * $moltiplicatore;  
  
    echo "5 * " . $moltiplicatore . " = " .  
$risultato . "<br/>";  
  
    $moltiplicatore++;  
  
}  
  
?>
```

Come nel ciclo for, anche con while l'esecuzione dello script termina quando la condizione (tra parentesi) diventa falsa. Fino a quel momento tutto quello ricompreso nelle parentesi graffe viene ripetuto.

do while

Ancora deve essere ricordato il ciclo **do while** che ha un funzionamento simile a quello di while con la sola differenza che la condizione è posta alla fine e non all'inizio. Torniamo al nostro esempio e vediamo come scriverlo con do while:

```
<?php  
$moltiplicatore = 1;  
do {
```

```
$risultato = 5 * $moltiplicatore;
```

```
echo "5 * " . $moltiplicatore . " = " .  
$risultato . "<br/>";
```

```
$moltiplicatore++;} while  
($moltiplicatore <= 10);
```

foreach

Per finire la nostra lezione sui cicli è opportuno ricordare l'esistenza di un ulteriore tipo di ciclo che viene utilizzato insieme alle array (vedremo dopo casa sono e come funzionano); si tratta del ciclo **foreach**.

Lo scopo di foreach è quello di costruire un ciclo che viene ripetuto per ogni elemento dell'array che gli passiamo... tuttavia per questo particolare tipo di

ciclo rimando alla lezione (che seguirà) sugli array.

Gli Array

Una array è una specie di "super-variabile" contenente una pluralità di valori invece di uno solo. Ma facciamo subito un paio esempi che valgono più di tante parole.

Poniamo di voler scrivere una sorta di lista di amici utilizzando, appunto, un array.

Uno dei modi per **scrivere il nostro array** è il seguente:

```
<?php
```

```
$amici = array("Luca", "Jacopo",  
"Felice", "Peppo");
```

```
?>
```

In sostanza abbiamo dato un nome al nostro array ed abbiamo inserito i vari elementi tra parentesi (dopo l'indicazione di "array") divisi da una virgola.

PHP associa automaticamente a ciascuno dei valori che abbiamo elencato un indice numerico, a partire da 0. Quindi, in questo caso, "Luca" assumerà l'indice 0, "Jacopo" l'indice 1, e così via.

Il risultato sarebbe stato il medesimo se avessimo scritto il nostro array in questa maniera:

```
<?php
```

```
$amici = array();  
$amici[0] = "Luca";  
$amici[1] = "Jacopo";  
$amici[2] = "Felice";  
$amici[3] = "Peppo";  
?>
```

Come è evidente nel primo modo l'indice dei vari *item* dell'array veniva assegnato in automatico da PHP, mentre in questo secondo caso lo abbiamo esplicitato noi.

Nel nostro esempio abbiamo utilizzato un array con indice numerico, tuttavia è opportuno ricordare che l'indice può essere anche di tipo stringa. Facciamo

un esempio di un array PHP contenente i dati di un ipotetico cliente in cui, appunto, l'indice degli elementi del nostro array è una stringa:

```
<?php  
$cliente["azienda"] = "Microsoft";  
$cliente["nome"] = "Bill";  
$cliente["cognome"] = "Gates";  
?>
```

Come avrete notato quando l'indice è di tipo stringa si usano gli apici anche all'interno della parentesi quadra (la regola è sempre la stessa: per i numeri non servono le virgolette, per le stringhe sì!).

Lavorare con gli array in PHP

Per riferirsi ad un singolo elemento dell'array si indica il nome dell'array seguito dall'indice contenuto fra parentesi quadre:

```
echo "Ciao " . $amici[1]; //output "Ciao  
Jacopo"
```

Volendo è anche possibile alterare l'array inizialmente definita sostituendo elementi o aggiungendone di nuovi.

Aggiungere un elemento ad un array

Per aggiungere un nuovo elemento alla nostra array (elemento che verrà

posizionato in fondo agli altri già presenti) si usa:

```
$amici[] = "Daniele"; //questa linea aggiunge l'elemento "Daniele" al nostro array
```

Nello specifico del nostro esempio, questo nuovo elemento verrà posizionato in coda al nostro array ed assumerà l'indice 4. PHP, infatti, quando si trova di fronte un'istruzione di quel tipo (parentesi quadre vuote), va a cercare l'elemento dell'array con l'indice più alto e lo aumenta di 1 per creare quello nuovo.

Sostituire un elemento di un array

Qualora volessimo **sostituire uno degli**

elementi già presenti useremo il nome del nostro array accompagnato dall'indice dell'elemento da sostituire tra le parentesi quadre. Poniamo ad esempio di voler sostituire l'elemento "Felice" con un nuovo elemento "Marcello". Ecco come fare:

```
$amici[2] = "Marcello"; //questa linea  
sostituisce l'elemento "Felice" con  
"Marcello"
```

Eliminare un elemento di un array

Per eliminare un elemento di un array PHP si utilizza la funzione **unset()** in questo modo:

```
unset($amici[2]);
```

Così facendo verrà eliminato l'elemento

con chiave 2 all'interno della nostra array `$amici`.

La funzione `count()`

Nel nostro esempio abbiamo usato un array di dimensioni molto ridotte, ma in realtà capita spesso di avere di fronte array molto lunghe e può essere utile conoscere il numero degli elementi che la compongono, in questo caso soccorre la **funzione `count()`** di PHP che useremo così:

```
$quanti_amici = count($amici);
```

Ciclare il contenuto di un array

Per stampare a video tutti gli elementi di un array PHP possiamo ricorrere ad un ciclo: ad esempio potremmo utilizzare

u n **ciclo for()** utilizzando appunto la funzione *count()* vista poco sopra per impostarne il limite finale:

```
<?php
```

```
$amici = array("Luca", "Jacopo",  
"Felice", "Peppo", "Gino", "Mario",  
"Antonio", "Roberto", "Massimo",  
"Giuseppe", "Matteo", "Silvio",  
"Michele", "Franco", "Guido", "Piero");
```

```
$quanti_amici = count($amici);
```

```
for ($i=0; $i<$quanti_amici; $i++)
```

```
{
```

```
    echo $amici[$i] . "<br/>";
```

```
}
```

?>

Come abbiamo già accenato nella lezione sui cicli, tuttavia, le array prevedono un tipo speciale di ciclo: il **ciclo foreach** che, sotto certi versi, risulta decisamente più semplice di *for()* per stampare il contenuto di un vettore.

Riprendiamo la nostra array \$amici dei precedenti esempi e costruiamo un ciclo usando, appunto, *foreach()*:

```
<?php
```

```
$amici = array("Luca", "Jacopo",  
"Felice", "Peppo", "Gino", "Mario",  
"Antonio", "Roberto", "Massimo",  
"Giuseppe", "Matteo", "Silvio",
```

```
"Michele", "Franco", "Guido", "Piero");  
foreach($amici as $amico)  
{  
    echo $amico . "<br/>";  
}  
?>
```

Il grosso vantaggio di `foreach` rispetto a `for()` è che non serve contare il numero degli elementi presenti in quanto PHP lo fa automaticamente. Per il resto il funzionamento è analogo a quello visto negli altri tipi di ciclo.

Le variabili GET e

POST

Una delle principali possibilità offerte dai linguaggi di scripting lato server è quella di generare contenuti (dinamicamente) sulla base delle richieste degli utenti. Questa interattività si realizza anche attraverso le variabili GET e POST che consentono, appunto, agli utenti di passare al server le loro richieste o preferenze attraverso i form (i classici moduli html) o semplici QueryString.

In pratica, attraverso GET e POST è possibile "raccogliere" gli input degli utenti i quali possono essere utilizzati per "indirizzare" il comportamento dei nostri script. Esistono, infatti, due

diversi modi per passare dati al server: il metodo POST (generalmente usato nei form) ed il metodo GET (generalmente usato nelle QueryString). Vediamoli separatamente.

La variabile `$_GET`

Con il **metodo GET** i dati vengono passati direttamente all'interno dell'indirizzo web (URL) della pagina, il quale si presenterà accompagnato da un punto di domanda (?) seguito dai dati organizzati in coppie nome/valore (qualora vi siano diverse coppie queste saranno legate tra loro dal simbolo &). Facciamo un esempio di utilizzo del metodo GET realizzando una semplice QueryString:


```
http://www.sito.com/automobili.php?  
marca=fiat&modello=panda
```

Nell'esempio qui sopra abbiamo ipotizzato di avere una pagina dinamica chiamata "automobili.php" in grado di visualizzare informazioni relativamente a diverse autovetture sulla base di due parametri (marca e modello).

Il contenuto della pagina "automobili.php" cambierà, ovviamente, ogni volta che varieranno i valori dei parametri marca e modello.

PHP memorizza i dati passati attraverso la QueryString all'interno della variabile `$_GET`. Nello specifico del nostro esempio faremo così per recuperarne il valore all'interno della pagina "automobili.php":

```
<?php
```

```
//Recupero il valore del parametro  
"tipo"
```

```
$marca_auto = $_GET['marca'];
```

```
//Recupero il valore del parametro  
"modello"
```

```
$modello_auto = $_GET['modello'];
```

```
//Ora stampo semplicemente a video il  
risultato
```

```
echo "Hai scelto una " . $marca_auto . "  
modello " . $modello_auto;
```

?>

Come avrete notato, per recuperare i dati passati dalla QueryString ho usato `$_GET` accompagnato da parentesi quadre al cui interno ho scritto (tra gli apici) il nome del parametro da recuperare (come vedremo tra breve useremo la stessa sintassi anche con `$_POST`). Da questa sintassi possiamo intuire che `$_GET` (come `$_POST`) è un **array superglobale** che, come tale, può essere richiamata in qualunque punto del nostro codice, anche all'interno di funzioni e metodi.

La variabile `$_POST`

Il **metodo POST** viene utilizzato per inviare i dati ad una applicazione PHP

tramite i form (moduli html). Prima di parlare nello specifico della variabile `$_POST` conviene fare un piccolo ripasso e vedere (brevemente) come funzionano i [form HTML](#).

il tag `<form>` viene sempre accompagnato da due attributi fondamentali - "method" e "action" - vediamo a cosa servono:

- l'attributo "method" determina, appunto, il metodo con cui i dati saranno inviati al server; può avere come valore sia GET (che genera una QueryString) che POST;
- l'attributo "action" ha come valore il percorso dell'applicazione a cui saranno inviati i dati.

Facciamo un esempio:

```
<form method="post"
action="applicazione.php">
```

```
Tuo Nome: <input type="text"
name="nome">
```

```
<input type="submit" name="submit"
value="invia">
```

```
</form>
```

Diversamente dal metodo GET, il metodo POST spedisce i dati in maniera non direttamente visibile per l'utente, attraverso la richiesta HTTP che il browser invia al server.

Tornando all'esempio visto sopra (il form HTML), per recuperare il valore del campo "nome" all'interno della nostra applicazione PHP useremo la

variabile `$_POST`. Ecco il codice del file "applicazione.php" cui punta il nostro form:

```
<?php
//Recupero il valore del parametro
"nome"

$nome_utente = $_POST['nome'];

//Ora stampo semplicemente a video il
risultato

echo "Ciao " . $nome_utente;

?>
```

register_globals

Settando l'impostazione "register_globals" su "php.ini" sarebbe possibile anche recuperare i dati in maniera più semplice. Infatti, se "register_globals" è "on" potremo recuperare i dati da form e querystring semplicemente utilizzando il nome del campo preceduto dal simbolo del dollaro.

Nell'esempio di prima, ad esempio, sarebbe stato sufficiente scrivere *\$nome* per recuperare il valore del relativo campo del form! Si noti, tuttavia, che per ragioni di sicurezza è bene mantenere su off lo stato di questa impostazione.

Il vecchio

\$HTTP_GET_VARS...

Gli array superglobali `$_GET` e `$_POST` sono stati introdotti nella versione 4.1.0 di PHP. In precedenza venivano utilizzati i corrispondenti `$HTTP_GET_VARS` e `$HTTP_POST_VARS`. Questi array sono disponibili anche nelle versioni attuali di PHP, ma il loro uso è sconsigliato, ed è presumibile che in futuro scompariranno del tutto.

Gestire i Cookie

I **cookie** sono un metodo rapido per memorizzare, sul computer dei nostri

utenti, delle informazioni che vogliamo persistano anche nelle successive visite al nostro sito.

I cookie sono molto utili per memorizzare piccoli dati come ad esempio il nome dell'utente o una serie di preferenze di navigazione. I cookie non sono adatti per informazioni critiche come password o dati personali in quanto potrebbero crearsi dei problemi di sicurezza.

Creare un cookie con PHP

Per **creare un cookie con PHP** useremo la **funzione setcookie()** in questo modo:

```
setcookie("nome_utente", "pippo",  
time()+3600);
```

Come vedete all'interno della funzione

setcookie() abbiamo inserito 3 parametri:

1. il primo specifica il nome identificativo del nostro cookie;
2. il secondo specifica il valore del cookie;
3. il terzo imposta la scadenza del cookie; se non impostiamo una data di scadenza il cookie non scadrà;

Perchè la funzione abbia esito positivo è **necessario inviare il cookie prima di ogni output**(esattamente come visto nella lezione precedente dedicata alla funzione header() di PHP). In caso contrario otterremo un errore. Semplificando:

```
// corretto
```

```
setcookie(...);
```

```
echo "...";
```

```
// errore
```

```
echo "...";
```

```
setcookie(...);
```

Facciamo ora un esempio su come memorizzare il nome di un nostro utente (richiesto tramite un form) all'interno di in un cookie. Ecco il codice:

```
<?php
```

```
//recupero il nome dal form
```

```
$nome = $_POST['nome'];
```

```
//memorizzo il nome in un cookie ed imposto la scadenza tra un'ora...
```

```
setcookie("nome_utente", $nome, time()+3600);
```

```
?>
```

Ora che abbiamo memorizzato nel cookie il nome dell'utente potremo tranquillamente richiamarlo in tutte le nostre pagine PHP in questo modo:

```
<?php
```

```
//recupero il valore del cookie...
```

```
$nome = $_COOKIE['nome_utente'];
```

```
//stampo a video il nome...
```

```
echo $nome;
```

```
?>
```

Verificare se un cookie esiste

E' bene precisare che la funzione `setcookie()` non garantisce la creazione del cookie voluto. La funzione, infatti, si limita ad inviare le giuste intestazioni HTTP, poi spetta al client del vostro utente accettarle o meno. Quindi, prima di utilizzare un cookie che si presume esistere, è buona norma effettuare una verifica di questo tipo:

```
<?php
```

```
// verifico se il cookie esiste
```

```
if (isset($_COOKIE['nome_utente'])) {
```

```
$nome = $_COOKIE['nome_utente'];  
  
echo $nome;  
  
}  
  
?>
```

Per effettuare questo tipo di controllo abbiamo utilizzato la [funzione isset\(\) di PHP](#) che verifica, appunto, se una variabile è stata settata o meno.

Cambiare il valore di un cookie

Se vogliamo **cambiare il valore del nostro cookie** basterà ripetere semplicemente l'operazione di assegnazione:

```
<?php
```

```
//imposto come valore "pippo"  
setcookie("nome_utente", "pippo");  
  
//ho cambiato idea e imposto come  
"pluto"  
setcookie("nome_utente", "pluto");  
?>
```

Cancellare un cookie

Se invece vogliamo **cancellare** il **cookie** basterà richiamare il cookie specificando *null* come valore:

```
setcookie("nome_utente", null);
```

oppure possiamo reimpostare la scadenza ad un momento passato:

```
setcookie("nome_utente", null, time()-3600);
```

In entrambi i casi il cookie verrà cancellato.

Le Sessioni

Le sessioni vengono utilizzate, ad esempio, nella gestione delle autenticazioni (login) degli utenti che, una volta loggati, verranno identificati come tali da tutte le pagine (.php) del sito.

La prima cosa da fare se vogliamo lavorare con le sessioni è impostare nel file di configurazione del PHP

("php.ini") la direttiva `session.save_path`, indicando la directory nella quale verranno salvate le informazioni sulle sessioni dei nostri utenti (se avete un sito in hosting non dovete fare nulla in quanto questo tipo di configurazione è già stato effettuato dal vostro provider di servizi).

La funzione da utilizzare all'interno delle nostre pagine PHP per aprire una sessione è **`session_start()`**. Questa funzione non prevede parametri.

La funzione `session_start()` deve essere necessariamente utilizzata prima dell'invio di output: nella parte precedente del nostro file `.php` non deve pertanto essere già stato scritto ed inviato del codice HTML (o altro tipo di

output) il quale comprometterebbe il buon esito della nostra funzione.

Esempio di utilizzo delle sessioni con PHP

Facciamo un esempio per vedere, in concreto, come funziona `session_start()`. Poniamo di voler aprire una sessione dove salvare username e password del nostro utente (dati che ci sono stati forniti tramite un form di login). Ecco il codice:

```
<?php
//Apro la sessione e...
session_start();
```

```
//Recupero username e password dal form
```

```
$username = $_POST['user'];
```

```
$password = $_POST['pass'];
```

```
//Salvo i dati...
```

```
$_SESSION['username'] = $username;
```

```
$_SESSION['password'] = $password;
```

```
?>
```

A questo punto abbiamo salvato all'interno della nostra sessione (grazie alla variabile superglobale **\$_SESSION**) due diversi

valori: username e password.

Nelle successive pagine .php sarà molto semplice recuperare questi valori, ecco un esempio:

```
<?php
```

```
//Apro la sessione e...
```

```
session_start();
```

```
//Recupero i dati...
```

```
$username = $_SESSION['username'];
```

```
$password = $_SESSION['password'];
```

```
//facciamo una stampata a video!
```

```
echo "Ciao " . $username . " la tua  
password è " . $password;
```

```
?>
```

E' bene precisare che la variabile superglobale `$_SESSION` non è altro che una semplice array e, come tale, può essere gestita. Se, ad esempio, vogliamo vedere tutti i valori salvati nella session potremo effettuare:

```
print_r($_SESSION);
```

Eliminare una sessione di PHP

Per finire vediamo come **eliminare una sessione** (ad esempio a seguito di logout).

```
//Per eliminare una specifica variabile  
di sessione useremo:
```

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

```
$_SESSION = array();
```

Gestire le intestazioni: la funzione `header()` di PHP

na funzione particolarmente utile ed interessante del **PHP** è sicuramente **`header()`**. Grazie a questa funzione, infatti, è possibile aggiungere delle intestazioni aggizionali a quelle normalmente scambiate tra il server ed il client.

Cosa sono gli headers

Gli *headers* (o "intestazioni" in italiano) sono delle meta-informazioni che vengono scambiate tra il server ed il client. Queste informazioni, che sono totalmente invisibili per l'utente (nel senso che non producono alcun output), possono essere indispensabili (nel senso che in mancanza la comunicazione non potrebbe funzionare correttamente) oppure puramente informative.

Tra queste informazioni vi sono, ad esempio, l'indicazione della risorsa richiesta e del protocollo utilizzato, le informazioni circa l'encoding ed il charset del documento, indicazione dello user-agent, eventuale referrer, ecc.

Vediamo un esempio di headers scambiati preliminarmente alla visualizzazione di una pagina web come questa:

```
GET /php/funzione-header_11798.html  
HTTP/1.1
```

```
Host: www.mrwebmaster.it
```

```
User-Agent: Mozilla /5.0 (Compatible  
MSIE 9.0;Windows NT 6.1;WOW64;  
Trident/5.0)
```

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Encoding: gzip
```

```
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
```


Cache-Control: no-cache

Gli headers, come potete vedere, sono formattati in questo modo:

NomeHeader: Valore (CRLF)

Non è previsto un ordine arbitrario per gli headers, quindi non è importante in quale ordine vengono inviati.

Diverse tipologie di headers

Gli headers possono essere suddivisi in quattro tipologie:

- General Headers (contengono informazioni generali come, ad esempio, lo stato della cache)
- Request Headers (sono richieste inviate dal browser)

- Response Headers (sono le risposte del server)
- Entity Headers (contengono meta-informazioni sull'entità - file - trasferito)

Inviare headers con PHP

Come abbiamo detto, tramite PHP è possibile arricchire lo scambio di informazioni tra il server ed il client, aggiungendo ulteriori meta-informazioni settate arbitrariamente dallo sviluppatore.

Lavorando con la funzione `headers()`, tuttavia, è bene tenere a mente una caratteristica molto importante degli headers: le intestazioni, infatti, devono essere inviate necessariamente prima

dell'output. In caso contrario PHP restituirà un errore di questo tipo:

```
Cannot modify header information -  
headers already sent by...
```

In poche parole, la definizione degli headers deve avvenire prima dell'invio di un qualsiasi altro dato attraverso, ad esempio, echo o print o attraverso l'inclusione di un qualche file HTML.

Alcuni esempi di utilizzo della funzione headers() di PHP

Grazie a questa funzione possiamo fare cose molto interessanti. Vediamo di seguito, molto brevemente, alcune

casiste piuttosto comuni:

Gestire redirect con PHP

Ad esempio possiamo effettuare un reindirizzamento dell'utente (cioè trasferirlo automaticamente da una pagina all'altra).

```
header('Location:  
http://www.sito.it/nuova-pagina.html');
```

E' anche possibile accompagnare il redirect con un messaggio di stato HTTP, ad esempio:

```
// pagina web spostata in modo  
permanente
```

```
header('HTTP/1.1 301 Moved  
Permanently');
```

```
// redirect dell'utente
```

```
header('Location:  
http://www.sito.it/nuova-pagina.html');
```

Per approfondire l'argomento vi suggerisco di dare una lettura a [questo articolo](#) dedicato, appunto, ai redirect con PHP.

Disabilitare la cache del browser

Attraverso l'invio delle giuste intestazioni è possibile disabilitare la cache del browser ed evitare che i contenuti del nostro sito vengano memorizzati in locale sul computer dell'utente. Una simile scelta diventa necessaria quando il contenuto di una

pagina è soggetto a continui cambiamenti. Vediamo un esempio di codice PHP:

```
header('Cache-Control: no-cache, no-store, max-age=0, must-revalidate');
```

```
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
```

```
header('Pragma: no-cache');
```

Per maggiori informazioni vi invito a leggere questo articolo che spiega come [disabilitare la cache del browser](#) con PHP.

Specificare un Content-Type

Oppure possiamo specificare un content-type conforme al tipo di output che vogliamo produrre (si pensi, ad

esempio, alla produzione dinamica di immagini attraverso l'uso di codice PHP). In questo caso il codice da usare sarà qualcosa del genere:

```
// Content-Type per immagini JPG
```

```
header('Content-Type: image/jpeg');
```

```
// Content-Type per immagini PNG
```

```
header('Content-type: image/png');
```

```
// Content-Type per immagini GIF
```

```
header('Content-Type: image/gif');
```

Ovviamente questo non riguarda solo le

immagini ma ogni altro tipo di file come, ad esempio, archivi compressi, file audio, video, ecc.

Creare pagine di errore dinamiche

E' possibile inviare anche messaggi di stato HTTP, cosa molto utile quando si gestiscono dinamicamente le pagine di errore. Vediamo qualche esempio:

```
// pagina non trovata
```

```
header('HTTP/1.1 404 Not Found');
```

```
// accesso vietato alla risorsa
```

```
header('HTTP/1.1 403 Forbidden');
```



```
// errore interno del server
```

```
header('HTTP/1.1 500 Internal Server  
Error');
```

Forzare il download di un file

Attraverso la funzione `header()` possiamo dire al browser di scaricare un certo documento invece di aprirlo:

```
header('Content-Type: application/octet-  
stream');
```

```
header('Content-Disposition: attachment;  
filename="brano.mp3");
```

```
header('Content-Transfer-Encoding:  
binary');
```

```
readfile('brano.mp3');
```

Per maggiori informazioni vi invito a leggere questo [breve articolo](#).

Conclusioni

Grazie a questa funzione di PHP è possibile intervenire su aspetti non visuali ma molto importanti di un'applicazione. Conoscere il funzionamento delle intestazioni e saper utilizzare correttamente la funzione `header()`, pertanto, è indispensabile per chiunque desideri sviluppare progetti web.

Funzioni in PHP

Una funzione è una porzione di codice caratterizzata da una **certa indipendenza** ed in grado di assolvere un **compito specifico** all'interno di un programma.

Mediante una funzione, di solito, si assolve un compito ben preciso che solitamente necessita di essere ripetuto più volte all'interno del programma. Così facendo, infatti, si snellisce il codice dell'applicazione e viene a semplificarsi il lavoro dello sviluppatore.

Per fare un esempio si pensi al calcolo dell'IVA su un prezzo: la procedura per svolgere tale calcolo matematico è

sempre la stessa e potrebbe essere richiesta in più punti di un programma. In circostanze come questa, quindi, la soluzione più logica è quella di incorporare la relativa *procedura* all'interno di una funzione che potrà poi essere richiamata ogni volta che ce ne sarà bisogno.

Funzioni native vs. funzioni proprie

Una funzione si dice "nativa" quando è incorporata nel linguaggio; si dice "propria" o "personalizzata" quando è definita dal programmatore all'interno del codice di un programma.

Nel corso della nostra guida abbiamo già incontrato diverse funzioni native di

PHP come, ad esempio, la funzione `header()` e la funzione `cookie()` ma ovviamente ne esistono diverse centinaia per gli scopi più diversi (alcune delle più importanti funzioni native di PHP le vedremo nelle prossime lezioni).

Creare una funzione in PHP

Una funzione personalizzata in PHP è definita mediante la *keyword* **function** seguita dal nome della funzione e dall'elenco di eventuali parametri richiesti racchiusi tra parentesi tonda. Di seguito la sintassi per la creazione di una funzione:

```
function nome_funzione($arg1,$arg2,...)
{
```

```
// ...  
  
// ...  
  
// ...  
  
}
```

Si noti che è anche possibile creare funzioni senza alcun parametro, in tal caso le parentesi tonde dopo il nome resteranno vuote:

```
function nome_funzione() {  
  
    // ...  
  
    // ... // ... }  
}
```

Gli argomenti elencati tra parentesi tonde sono obbligatori in quanto funzionali all'esecuzione della

procedura delegata alla funzione. Tuttavia è anche possibile prevedere argomenti facoltativi semplicemente prevedendo un valore di *default* (cioè un valore da utilizzare nel caso in cui non ne venga settato un altro):

```
function
nome_funzione($arg1,$arg2,$arg3=false)
{
    // ...
    // ...
    // ...
}
```

```
// utilizzo
```

```
$var = nome_funzione(10,8);
```

```
// oppure
```

```
$var = nome_funzione(10,8,true);
```

Nell'esempio visto qui sopra la funzione ammette tre argomenti dei quali i primi due sono obbligatori mentre il terzo è facoltativo e qualora venga omesso avrà come valore *false*.

All'interno della nostra funzione (dove abbiamo messo i puntini, per intenderci) andremo scrivere una serie di istruzioni PHP sulla base delle regole e delle logiche che abbiamo visto sino ad ora

all'interno della nostra guida: potremo usare variabili, strutture condizionali, cicli e quant'altro ci verrà utile.

E' importante sottolineare, tuttavia, che all'interno della nostra funzione avremo a disposizione tutte le variabili ricevute in argomento e le variabili superglobali (come, ad esempio, `$_GET`, `$_POST`, `$_SESSION`, ecc.) mentre le variabili definite all'interno del nostro programma (ma esternamente alla funzione) non saranno accessibili se non esplicitamente "importate".

Per utilizzare variabili esterne alla funzione si utilizza l'istruzione **global** seguita dal nome delle variabili (separate da virgola) che si desidera utilizzare all'interno della

funzione. Vediamo un esempio:

```
$nome = 'Mario';
```

```
$cognome = 'Rossi';
```

```
function scrivi_nome() {  
    global $nome, $cognome;  
    return $nome . ' ' . $cognome;  
}
```

```
echo scrivi_nome();
```

Nell'esempio appena visto abbiamo incontrato l'istruzione **return** che è utilizzata per restituire il valore

"calcolato" dalla nostra funzione. Utilizzando *return* il valore restituito all'interno del programma può essere memorizzato all'interno di una variabile. Tornando all'esempio visto poco sopra avremmo potuto fare:

```
$nome_completo = scrivi_nome();
```

E' bene precisare che quando si usa *return* si esce dalla funzione quindi, eventuale codice successivo ad un *return* non verrà elaborato. Vediamo un esempio:

```
function divisione($n1,$n2) {  
    if ($n2 == 0) return 'Non puoi dividere  
per zero!';  
  
    return ($n1/$n2);  
}
```

```
}  
  
// stampo il risultato  
echo divisione(10/0);
```

Nell'esempio visto sopra non è stato necessario utilizzare un *else* in quanto il verificarsi della condizione ha prodotto il `return` che chiude la funzione. L'utilizzo di *return*, tuttavia, non è obbligatorio: se il risultato dell'elaborazione non necessita di essere incapsulato in una variabile o nuovamente elaborato, nulla vieta di utilizzare un semplice *echo* (o altro comando o istruzione):

```
function divisione($n1,$n2) {
```

```
if ($n2 == 0) echo 'Non puoi dividere
per zero!';

else echo ($n1/$n2);

}

// stampo il risultato

divisione(10/0);
```

Richiamare una funzione

Come abbiamo già visto nei precedenti esempi per utilizzare una funzione è sufficiente richiamarne il nome indicando, ove necessario, i parametri richiesti in argomento. Ad esempio:

```
// funzione senza argomenti e con valore
```

di ritorno

```
$var = nome_funzione();
```

```
// funzione con argomenti e con valore di ritorno
```

```
$var = nome_funzione($arg1,$arg2);
```

```
// funzione senza argomenti e senza valore di ritorno
```

```
nome_funzione();
```

```
// funzione con argomenti e senza valore
```

di ritorno

```
nome_funzione($arg1,$arg2);
```

Un esempio completo: una funzione per calcolare il prezzo con IVA

Per concludere questa lezione vediamo un esempio completo di funzione PHP personalizzata. Vediamo di seguito una funzione per il calcolo del prezzo + IVA:

```
function prezzo_ivato($prezzo) {  
  
    $aliquota = 22;  
  
    $prezzo_ivato = ($prezzo*  
(100+$aliquota))/100;
```

```
return  
number_format($prezzo_ivato,2,',','');  
}  
  
// utilizzo la funzione  
  
$prezzo = 70;  
  
$prezzo_finale = prezzo_ivato($prezzo);  
  
echo 'Il prezzo finale è di ' .  
$prezzo_finale . ' Euro';
```

Nella lezione precedente (dedicate alle funzioni PHP) abbiamo visto la differenza tra funzioni "native" e "personalizzate". A partire da questa

questa lezione passeremo in rassegna alcune delle (tantissime) **funzioni native offerte da PHP** per svolgere le più diverse operazioni.

In questa lezione vedremo le principali funzioni per la gestione delle variabili.

empty

Questa funzione verifica se una variabile è vuota oppure no. Una variabile è definita "vuota" se:

- non esiste;
- contiene una stringa vuota;
- contiene un valore numerico pari a 0 (equivalente a 0.0 o "0");
- è un array() senza elementi;
- è FALSE o NULL

La funzione `empty()` restituisce `true`

(Vero) o false (Falso).

```
$var = 0;  
  
if (empty($var)) {  
    echo 'la variabile è vuota';  
}  
else {  
    echo 'la variabile NON è vuota';  
}
```

isset

Questa funzione di PHP verifica se una variabile è stata definita oppure no. Restituisce true o false a seconda che la variabile sia stata definita o meno.

La funzione `isset()` restituisce false anche se la variabile ha valore NULL.

is_string e is_numeric

Queste due funzioni native di PHP, come lascia intuire il loro nome, sono utilizzate rispettivamente per verificare se una variabile è una stringa oppure un valore numerico. Restituiscono true o false a seconda che il controllo dia esito positivo o negativo.

```
$var = 123;  
  
if (is_string($var)) {  
    echo 'la variabile è una stringa';  
}  
else {  
    echo 'la variabile NON è una  
stringa';  
}
```

```
}
```

Il funzionamento di `is_numeric()` è identico.

`is_int` e `is_float`

Queste due funzioni PHP sono utilizzate per la verifica di variabili numeriche e, più precisamente, il loro scopo è di verificare se si tratta di un numero intero (`is_int`) o di un numero decimale (`is_float`).

`is_array`, `is_bool` e `is_null`

Queste funzioni, come quelle viste più sopra, servono per verificare il tipo di valore contenuto in una variabile, più

precisamente:

- `is_array` - verifica se una variabile è un array;
- `is_bool` - verifica se una variabile contiene un valore booleano (`true/false`);
- `is_null` - verifica se una variabile è `NULL`.

gettype

La funzione `gettype()` di PHP restituisce il tipo della variabile presa in argomento. I possibili valori restituiti sono:

- `boolean`
- `integer`
- `double`
- `string`

- array
- object
- resource
- NULL
- unknown type

```
$var = 123;
```

```
echo gettype($var);
```

Il nostro esempio restituisce: integer

Funzioni PHP per la gestione delle stringhe

echo e print

Queste due funzioni (già viste più volte

nel corso della nostra guida) servono per stampare a video il contenuto di una variabile, un numero o una stringa di testo. Esempi:

```
$var = 'Evviva Google';  
  
echo $var;  
  
print 123;
```

Queste due funzioni sono sostanzialmente equivalenti.

strlen

Restituisce un valore numerico corrispondente al numero di caratteri di cui è composta una stringa:

```
echo strlen('Younes');
```

Stamperà a video: 1

strrev

Questa funzione restituisce una stringa invertendo l'ordine di caratteri. Vediamo un esempio:

```
echo strrev('google');
```

Stamperà a video: elgoog

strtolower e strtoupper

Queste due funzioni servono rispettivamente a trasformare una stringa tutto in minuscolo o in maiuscolo. Vediamo degli esempi:

```
echo strtolower('Mr.Webmaster');
```

Stamperà a video: mr.webmaster

```
echo strtoupper('Mr.Webmaster');
```

Stamperà a video: MR.WEBMASTER

ucfirst e ucwords

Servono rispettivamente a trasformare in maiuscolo la prima lettera di una sola parola e di tutte le parole di una frase. Vediamo degli esempi:

```
echo ucfirst('evviva questo sito');
```

Stamperà a video: Evviva questo sito

```
echo ucwords('evviva questo sito');
```

Stamperà a video: Evviva Questo Sito

explode

Questa funzione serve per suddividere una stringa in più parti sulla base di un elemento separatore. Il risultato sarà una array composta dai diversi elementi estratti. Poniamo, ad esempio, di voler dividere la stringa "13-34-96" usando

come divisore il trattino (-):

```
$nums = explode('-', '13-34-96');
```

Il risultato sarà un array di tre elementi (13,34,96).

htmlspecialchars e strip_tags

Queste due funzioni di PHP rivestono un ruolo molto importante nella gestione delle stringhe e della sicurezza. In pratica, attraverso queste due funzioni, si impedisce di passare codice HTML il quale viene trasformato in entità (htmlspecialchars) o filtrato (strip_tags).

Vediamo degli esempi:

```
$var = 'Formatto il <b><i>testo</i></b>';
```

```
echo htmlspecialchars($var);
```

Il risultato a video sarà: Formato il **
<i>testo</i>**

```
echo strip_tags($var);
```

Il risultato a video sarà: Formato il
testo

Si noti che `strip_tags()` ammette anche un secondo parametro facoltativo con l'elenco di eventuali tag ammessi, ad esempio:

```
echo strip_tags($var,'<i>');
```

Il risultato a video sarà: Formato
il *testo*

str_replace

Consente di sostituire una sotto-stringa

all'interno di una stringa.

La funzione **str_replace** effettua la sostituzione di tutte le occorrenze di una stringa all'interno di un'altra stringa. La sintassi è la seguente:

```
str_replace(cerca, sostituisci  
,dove_cercare)
```

Vediamo un esempio:

```
echo str_replace("mela", "pera", "Paolo  
mangia la mela");
```

l'output prodotto sarà:

```
Paolo mangia la pera
```

La funzione **str_replace** può accettare anche argomenti di tipo Array. Vediamo un esempio:

```
$stesto = "I miei amici Luca, Paolo e  
Claudio sono al mare";
```

```
$cerca =  
array("Luca","Paolo","Claudio");  
$sostituisci =  
array("Max","Roberto","Alex");  
echo str_replace($cerca, $sostituisci,  
$testo);
```

L'output restituito sarà:

I miei amici Max, Roberto e Alex sono
al mare

Attenzione: la funzione in oggetto è *case sensitive* cioè distingue tra maiuscole e minuscole. Per effettuare sostituzioni **NON** *case sensitive* è possibile utilizzare l'analoga funzione [str_ireplace](#)

Gestire date e orari in PHP

In questo capitolo passeremo in rassegna alcune funzioni per la **gestione delle date in PHP** concentrando la nostra attenzione, in particolare, sulla funzione **date()** al fine di capire come poterla usare nei nostri script. Le funzioni per la manipolazione di date ed orari sono determinanti nello sviluppo di applicazioni e devono essere approfondite con la dovuta diligenza.

Prima di passare in rassegna le principali funzioni PHP per la manipolazione di date e orari, tuttavia, è necessario fare una premessa ed

introdurre il concetto di **timestamp**. Il timestamp è un valore numerico corrispondente al numero di secondi trascorsi da un preciso momento storico che prende il nome di *epoch*. Questo momento storico coincide con il 1 gennaio 1970 alle ore 00:00.

Questa caratteristica è tipica dei sistemi UNIX sui quali, appunto, PHP è nato ed ha preso vita.

Questo modo di gestire le date, diversamente da come può apparire a chi è alle prime armi, è molto efficiente in quanto le date vengono trasformate in semplici numeri interi rendendo molto agevole ogni operazione di confronto o di aggiunta e sottrazione.

La funzione time

La funzione `time()` di PHP consente di conoscere il valore attuale del timestamp. In poche parole, questa funzione restituisce il numero di secondi trascorso dal famoso 1 gennaio 1970.

```
echo time();
```

Il risultato sarà qualcosa del genere:
1064390400

La funzione mktime

Questa funzione consente di risalire al timestamp partendo da una data arbitraria. La sintassi di `mktime` è la seguente:

```
mktime(ore, minuti, secondi, mese,  
giorno, anno);
```


Vediamo un esempio:

```
mktime(10, 0, 0, 9, 24, 2003);
```

Il risultato sarà: 1064390400

La funzione date

Questa funzione, come lascia intendere il suo nome, consente di **formattare una data ed un orario** sulla base delle impostazioni del server sul quale girano i nostri script. Per prima cosa vediamo un semplicissimo esempio:

```
echo date("l");
```

Nell'esempio qui sopra la nostra funzione `date()` riceve un unico parametro ("l") che, come vedremo, corrisponde alla richiesta di mostrare il nome del giorno della settimana

corrente.

Si noti che la funzione `date()` di PHP prevede due parametri: il primo (obbligatorio) è una stringa con il *formato* richiesto per l'output; il secondo (facoltativo) è il valore di *timestamp* che si desidera formattare (se omissso verrà preso in considerazione il *timestamp* attuale).

Vediamo un esempio completo:

```
echo date("d/m/Y", 1064390400);
```

```
// output: 24/09/2003
```

Vediamo ora i parametri attraverso i quali compilare la stringa per la formattazione dell'output:

- **a** - fa comparire accanto all'ora la dicitura "am" se l'ora è compresa tra

mezzanotte e mezzogiorno, fa invece comparire la dicitura "pm" se l'ora è compresa tra mezzogiorno e mezzanotte. Per esempio 9:25 am per le nove e venticinque del mattino, 9:25 pm per le ventuno e venticinque di sera.

- **A** - ha la stessa funzione del parametro precedente, ma fa comparire le lettere in maiuscolo anzichè in minuscolo. Per esempio 9:25 AM per le nove e venticinque del mattino, 9:25 PM per le ventuno e venticinque di sera.
- **D** - indica le prime tre lettere del nome inglese del giorno della settimana. Per esempio Sun, Mon, Fri, Sat.

- **I** - indica il nome inglese del giorno della settimana. Per esempio Sunday, Monday, Friday, Saturday.
- **g** - indica l'ora, in formato 12 ore (da usare magari con i parametri a e A), senza l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 1 e 12.
- **G** - indica l'ora, in formato 24 ore, senza l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 1 e 24.
- **h** - indica l'ora, in formato 12 ore, con l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 01 e 12.
- **H** - indica l'ora, in formato 24 ore, con l'eventuale zero iniziale. Quindi

assumerà un valore compreso tra 01 e 24.

- **i** - indica i minuti, con l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 00 e 59.
- **I** - restituisce 1 se c'è l'ora legale, 0 se c'è quella solare.
- **j** - indica il giorno del mese, senza l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 1 e 31.
- **d** - indica il giorno del mese, con l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 01 e 31.
- **L** - restituisce 1 se l'anno è bisestile, 0 se non lo è.
- **m** - indica il numero del mese con

l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 01 e 12.

- **n** - indica il numero del mese senza l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 1 e 12.
- **M** - indica le prime tre lettere del nome inglese del mese in cui ci troviamo. Per esempio Jan, Mar, Aug, Nov.
- **F** - indica il nome inglese per intero del mese in cui ci troviamo. Per esempio January, March, August, November.
- **O** - indica la differenza dal meridiano di Greenwich. Per esempio +0200 per indicare due ore

di ritardo, o -0400 per indicare quattro ore di anticipo.

- **Z** - indica la differenza dal meridiano di Greenwich in secondi. Per esempio +43200 o - 43200.
- **r** - restituisce la data formattata secondo la norma RFC 822. Per esempio Thu, 21 Dec 2000 16:01:07 +0200. (Disponibile a partire da PHP 4.0.4)
- **s** - indica i secondi, con l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 00 e 59.
- **S** - indica il suffisso inglese per i numeri cardinali. Per esempio st, nd, rd, th.
- **t** - indica il numero di giorni di un dato mese senza lo zero iniziale.

Quindi assumerà un valore compreso tra 1 e 31.

- **U** - indica il numero di secondi trascorsi dall'Unix Epoch, ovvero il 1 Gennaio 1970.
- **w** - indica il numero del giorno della settimana senza lo zero iniziale. Bisogna tenere conto che la settimana inglese inizia la Domenica, e che in PHP gli elementi si contano a partire da 0. Quindi assumerà 0 per Domenica, 1 per Lunedì e così via fino a 6 per Sabato.
- **W** - indica il numero della settimana in cui ci troviamo. Per questo parametro le settimane iniziano il Lunedì, e non la Domenica.
- **Y** - indica l'anno utilizzando quattro

cifre. Per esempio 2003.

- **y** - indica l'anno utilizzando le ultime due cifre. Per esempio 03.
- **z** - indica il numero di giorno dell'anno senza l'eventuale zero iniziale. Quindi assumerà un valore compreso tra 1 e 366

Esempi di utilizzo di `date()` in PHP

Ora che conosciamo la teoria vediamo qualche esempio pratico di utilizzo della funzione `date()` di PHP.

Immaginiamo di voler ottenere una data del tipo 04/09/03, dovremmo scrivere una riga come questa:

```
date("d/m/y");
```

Per ottenere una data del tipo 4-9-2003

dovremmo utilizzare:

```
date("j-n-Y");
```

Per ottenere un'orario del tipo 15:54:06
dovremmo utilizzare:

```
date("H:i:s");
```

Per ottenere un'orario del tipo 3.54 pm
dovremmo utilizzare:

```
date("g.i a");
```

Passiamo adesso a vedere come ottenere delle date un po' più complesse. Nulla infatti ci vieta di ottenere un output di data/orario del tipo 23/9/2003 9:54.65 PM. La riga da scrivere sarà come questa:

```
date("d/n/Y g:i.s A");
```

Caratteri speciali e backslash

Una nota importante: nel primo parametro (cioè nella stringa di formattazione) possiamo usare, come avete visto, anche altri caratteri rispetto a quelli "speciali" elencati sopra. E' bene ricordare, però, che se scriviamo "a" al suo posto PHP inserirà la dicitura am/pm... ma se non vogliamo che ciò accada? se vogliamo che venga scritta la lettera "a" come possiamo fare? Semplice... basta farla precedere da un backslash. Provate ad esempio ad eseguire questo codice:

```
date('d/m/Y \a\\|\e H:i:s');
```

Il risultato sarà: 04/09/2003 alle 15:54:06

La funzione checkdate

Grazie alla funzione `checkdate` di PHP, come lascia intendere il nome, è possibile verificare se una data è corretta oppure no. La sua sintassi è la seguente:

```
checkdate(mese, giorno, anno);
```

la funzione restituisce `true` o `false` a seconda che la verifica sia andata o buon fine oppure no.

Vediamo un esempio pratico:

```
if(checkdate(11,31,2003)) {  
    echo 'La data è corretta';  
}  
else {  
    echo 'La data è sbagliata';  
}
```

Nel nostro esempio, ovviamente, il risultato sarà negativo in quanto il mese di novembre ha 30 giorni e non 31.

Verificare file e cartelle con PHP: `is_file`, `is_dir`, `file_exists`

PHP è un linguaggio di scripting completo, tra le sue possibilità vi è anche quella di interagire con il

filesystem del server sul quale sta girando. Grazie a questa interazione è possibile lavorare con le cartelle e con i file sia in lettura che in scrittura. Le funzioni native che PHP offre per la manipolazione del filesystem sono molteplici, ma in questa sede ci limiteremo ad analizzare le fondamentali.

Nelle prossime lezioni vedremo come utilizzare PHPO per svolgere i compiti più comuni di interazione col filesystem, ma in questa lezione passeremo in rassegna le cosiddette funzioni di verifica cioè quelle funzioni che consentono allo sviluppatore di verificare se una data risorsa esiste, è un file oppure una cartella.

is_dir

Consente di verificare se un percorso esistente è una valida cartella

```
$path = 'foto';  
  
if (is_dir($path)) {  
    echo $path . ' è una cartella';  
}  
else {  
    echo $path . ' NON è una cartella';  
}
```

is_file

Consente di verificare se un percorso esistente è un file

```
$path = 'miofile.txt';
```

```
if (is_file($path)) {  
    echo $path . ' è un file';  
}else{  
    echo $path . ' NON è un file';  
}
```

file_exists

Consente di verificare se un dato file o una data cartella esiste

```
$path = 'miofile.txt';  
if (file_exists($path)) {  
    echo 'il file esiste';  
}else{
```



```
echo 'il file NON esiste';
```

```
}
```

Nelle prossimi capitoli della nostra guida a PHP vedremo come effettuare varie operazioni di interazione tra PHP ed il filesystem come, ad esempio, come creare file e cartelle sul server e come scriverci dentro.

Creare, cancellare e copiare file e cartelle con PHP

Un altro set di funzioni PHP molto

interessanti sono quelle che consentono di creare e cancellare file e directory.

touch

Grazie alla funzione touch è possibile creare un file specificandone il percorso ed il nome:

```
touch('file-da-creare.txt');
```

unlink

La funzione unlink() consente di cancellare un file e restituisce true o false a seconda che l'operazione abbia successo o meno:

```
$file = 'miofile.txt';
```

```
if (unlink($file)) {
```

```
echo 'il file è stato cancellato';  
}  
else {  
    echo 'il file NON è stato cancellato';  
}
```

copy

La funzione `copy()` consente di copiare un file e restituisce `true` o `false` a seconda che l'operazione abbia successo o meno. Questa funzione prevede due argomenti obbligatori: il file sorgente e quello di destinazione. Vediamo un esempio:

```
copy('miofile.txt','filecopiato.txt');
```

mkdir

Grazie a `mkdir()` è possibile creare una cartella. Restituisce `true` o `false` a seconda che l'operazione abbia successo o meno.

```
$path = 'cartella-da-creare';  
  
if(!mkdir($path)) {  
    echo 'La cartella è stata creata';  
}
```

In realtà la funzione in oggetto prevede anche altri parametri facoltativi, ma in questa sede li tralasciamo delegando a futuri approfondimenti.

`rmdir`

La funzione `rmdir()` di PHP è l'esatto contrario di `mkdir()`. La sua funzione è,

in pratica, quella di cancellare una cartella. Perchè la funzione abbia successo (e restituisca quindi true) è necessario che la cartella sia vuota. Non è possibile, infatti, cancellare cartelle piene.

```
$path = 'cartella-da-cancellare';  
if(!rmdir($path)) { echo 'NON è stato  
possibile cancellare la cartella';}
```

Grazie a **PHP**, come abbiamo visto, è piuttosto facile interagire col filesystem, in questa lezione della nostra guida vedremo le funzioni base per lavorare con i file, più precisamente vedremo come aprire un file di testo e come leggerne e modificarne il contenuto

fopen

La funzione di base per "aprire" un file è **fopen()**. Questa funzione restituisce "true" in caso di esito positivo o "false" in caso di esito negativo. Ecco come usare "fopen()".

```
$fp = fopen("data.txt", "r");
```

```
if(!$fp) die ("Errore nella operazione con  
il file");
```

Come vedete all'interno della funzione, oltre al percorso del file da aprire, abbiamo passato un secondo attributo "r" (che prende il nome di "mode" o modalità). Col secondo attributo abbiamo impostato la modalità di sola lettura ma avremmo potuto impostare anche altri valori, vediamoli di seguito:

- **r** - Apertura del file per sola lettura;
- **r+** - Apertura del file per lettura e scrittura;
- **w** - Apertura del file per sola scrittura; I contenuti del file esistente andranno persi, qualora il file non esiste PHP cercherà di crearlo;
- **w+** - Apertura del file per lettura e scrittura. I contenuti del file esistente andranno persi, qualora il file non esiste PHP cercherà di crearlo;
- **a** - Apertura del file per sola aggiunta. I nuovi dati verranno aggiunti in coda ai dati già presenti, qualora il file non esiste PHP cercherà di crearlo;
- **a+** - Apertura del file per lettura e aggiunta. I nuovi dati verranno

aggiunti in coda ai dati già presenti, qualora il file non esiste PHP cercherà di crearlo;

Dopo aver aperto il file possiamo finalmente lavorarci sopra. Le funzioni utili a questo punto saranno "fread()" e "fwrite()" che utilizzeremo, rispettivamente, per leggere e per scrivere il contenuto del file.

fread

Questa funzione viene utilizzata per estrarre una stringa di caratteri da un file. Quindi (dopo aver aperto il file con fopen() nel modo visto sopra) aggungeremo qualcosa del genere:

```
$data = fread($fp, 10);
```



```
echo $data;
```

All'interno della nostra funzione abbiamo passato due argomenti: il primo è il puntatore del file, il secondo è un valore numerico che sta ad indicare il numero massimo di byte da leggere (nell'esempio abbiamo messo 10). Arrivato a quel numero PHP smetterà di leggere. Con "echo" abbiamo poi chiesto al nostro script di stampare a video il contenuto ricavato dal file.

fwrite

Ovviamente, prima di usare questa funzione dovremo aprire il file in modo adeguato, quindi non utilizzeremo "r" ma, ad esempio, "w" in modo da consentire la scrittura. Dopodichè

andremo ad usare `fwrite()` in questo modo:

```
fwrite($fp, "ciao a tutti");
```

Il primo attributo della funzione, come per `fread()`, è il puntatore del file, il secondo, invece, è la stringa che andremo a scrivere all'interno del nostro file (la posizione in cui verrà inserita la stringa di testo all'interno del file dipende dal valore precisato per "mode" al momento dell'apertura del file)

`fclose`

Una volta concluso il nostro lavoro sul file è buona norma chiedere a PHP di chiudere lo stesso cancellando il puntatore. In questo modo:

```
fclose($fp)
```

Questa funzione richiede un solo attributo, cioè il nome del puntatore relativo al file che desideriamo chiudere.

La funzione mail() di PHP

Inviare **e-mail con PHP** è un'operazione abbastanza semplice: per spedire un messaggio di posta elettronica dalle pagine del nostro sito web, infatti, è sufficiente richiamare la **funzione mail()** la quale consente, appunto, di inviare email con codifica MIME.

La funzione mail(), una volta richiamata all'interno della nostra applicazione

PHP, "contatterà" il sistema postale del nostro server (sendmail o server SMTP) intimandogli di spedire una mail con le caratteristiche definite dallo sviluppatore. Ovviamente, nel caso in cui il nostro server non sia attrezzato di un sistema di spedizione attivo e funzionante (ad esempio perchè il componente è stato bloccato o la porta chiusa) la funzione mail() restituirà FALSE (restituirà TRUE in caso di successo).

Inviare una semplice mail con PHP

Questa la sintassi di base della funzione mail() di PHP:

```
mail($destinatario, $oggetto,  
$messaggio)
```

Solitamente, tuttavia, si utilizza anche un quarto parametro (facoltativo) per passare alla funzione i cosiddetti *headers*. Senza questo quarto parametro, infatti, le mail verrebbero spedite indicando come mittendere l'indirizzo di default del server con problemi sia dal punto di vista della deliverability (i messaggi potrebbero essere identificati come spam) che pratici (eventuali *reply* non andrebbero a buon fine).

La sintassi completa della nostra funzione `mail()`, quindi, è la seguente:

```
mail($destinatario, $oggetto,  
$messaggio, $headers)
```

Ecco un piccolo esempio di codice PHP per l'invio di una semplice e-mail di testo:

```
<?php  
  
// definisco mittente e destinatario della  
mail  
  
$nome_mittente = "Mio Nome";  
  
$mail_mittente = "mittente@sito.com";  
  
$mail_destinatario =  
"destinatario@sito.com";  
  
// definisco il subject ed il body della  
mail
```

```
$mail_oggetto = "Messaggio di prova";
```

```
$mail_corpo = "Questo è un messaggio  
di prova per testare la mia  
applicazione";
```

```
// aggiusto un po' le intestazioni della  
mail
```

```
// E' in questa sezione che deve essere  
definito il mittente (From)
```

```
// ed altri eventuali valori come Cc,  
Bcc, ReplyTo e X-Mailer
```

```
$mail_headers = "From: " .  
$nome_mittente . " <" . $mail_mittente .  
">\r\n";
```

```
$mail_headers .= "Reply-To: " .
```

```
$mail_mittente . "\r\n";
```

```
$mail_headers .= "X-Mailer: PHP/" .  
phpversion();
```

```
if (mail($mail_destinatario,  
$mail_oggetto, $mail_corpo,  
$mail_headers))
```

```
    echo "Messaggio inviato con successo  
a " . $mail_destinatario;
```

```
else
```

```
    echo "Errore. Nessun messaggio  
inviato.";
```

```
?>
```


Gli headers

Abbiamo visto che gli headers contengono una serie di informazioni supplementari. Nel nostro esempio:

- nome ed indirizzo email del mittente;
- indirizzo *reply-to* (rispondi a...) che nel nostro caso corrisponde al mittente (ma nulla vieta che sia un indirizzo differente);
- strumento utilizzato pr la spedizione (*X-Mailer*).

Si noti che tutte queste informazioni sono separate da dei ritorni a capo fisici ("`\r\n`") che non possono essere omessi.

Inviare una mail in HTML con PHP

Volendo è anche possibile spedire mail con formattazione HTML (nell'esempio visto sopra si trattava di una semplice mail *plain text*). Per inviare una mail in HTML utilizzando la funzione mail() di PHP sarà sufficiente modificare il codice visto sopra in questo modo:

```
<?php  
  
// definisco mittente e destinatario della  
mail  
  
$nome_mittente = "Mio Nome";  
  
$mail_mittente = "mittente@sito.com";  
  
$mail_destinatario =  
"destinatario@sito.com";
```

```
// definisco il subject
```

```
$mail_oggetto = "Messaggio di prova";
```

```
// definisco il messaggio formattato in  
HTML
```

```
$mail_corpo = <<<<HTML
```

```
<html>
```

```
<head>
```

```
  <title>Una semplice mail con PHP  
formattata in HTML</title>
```

```
</head>
```

```
<body>
```

Questo è un messaggio di prova
l'invio di mail in HTML con la
funzione mail() di PHP

```
</body>
```

```
</html>
```

```
HTML;
```

```
// aggiusto un po' le intestazioni della  
mail
```

```
// E' in questa sezione che deve essere  
definito il mittente (From)
```

```
// ed altri eventuali valori come Cc,  
Bcc, ReplyTo e X-Mailer
```

```
$mail_headers = "From: " .  
$nome_mittente . " <" . $mail_mittente .  
>\r\n";
```

```
$mail_headers .= "Reply-To: " .  
$mail_mittente . "\r\n";
```

```
$mail_headers .= "X-Mailer: PHP/" .  
phpversion() . "\r\n";
```

// Aggiungo alle intestazioni della mail
la definizione di MIME-Version,

// Content-type e charset (necessarie per
i contenuti in HTML)

```
$mail_headers .= "MIME-Version:  
1.0\r\n";
```

```
$mail_headers .= "Content-type:  
text/html; charset=iso-8859-1";
```

```
if (mail($mail_destinatario,  
$mail_oggetto, $mail_corpo,  
$mail_headers))
```

```
    echo "Messaggio inviato con successo  
a " . $mail_destinatario;
```

```
else
```

```
    echo "Errore. Nessun messaggio  
inviato.";
```

```
?>
```

Come avrete notato il codice non è molto differente da quello visto in

precedenza, le uniche differenze sono:

- il corpo del messaggio è formattato in HTML;
- negli *headers* abbiamo aggiunto l'indicazione della *MIME-Version*, ed il *Content-type* (text/html) e del *charset*.

Conclusioni

Ovviamente è possibile personalizzare i codici qui proposti nel modo che si ritiene più opportuno e sfruttare la funzione `mail()` di PHP per i più svariati utilizzi. Con qualche piccolo accorgimento, infatti, potrete creare facilmente applicazioni molto interessanti come, ad esempio, un formmail oppure un sistema per l'invio

di notifiche automatiche.

PHP e MySQL

In questa lezione vedremo insieme come

è possibile far interagire le nostre pagine PHP con i database MySQL. Per dovere di completezza non possiamo non ricordare che PHP è in grado di connettersi a diversi database server (MySQL, MS Access, PostgreSQL, Oracle, Microsoft Sql Server, Sybase,...) tuttavia noi ci limiteremo a vedere l'interazione con MySQL che è senza dubbio la soluzione più comune e diffusa.

Lavorare con PHP/MySQL

MySQL è un database veloce e potentissimo in grado di gestire applicazioni con un elevato grado di criticità e, cosa non secondaria, è un software open source, liberamente scaricabile dal sito www.mysql.com.

Come abbiamo accennato nella lezione precedente PHP mette a disposizione dello sviluppatore diverse funzioni per interagire con i database MySQL. Vediamo insieme le più importanti.

Connettersi ad un database MySQL

Per prima cosa vediamo come fa PHP a connettersi al MySQL Server. Allo scopo soccorre la funzione **mysql_connect()** che si utilizza con la seguente sintassi:

```
mysql_connect(server, utente,  
password);
```

Ad esempio:

```
$myconn = mysql_connect('localhost',  
'pippo', 'xxxxxx') or die('Errore...');
```

si noti l'utilizzo del comando *die()* il cui scopo è, in caso di errore, bloccare l'elaborazione dello script e stampare a video un messaggio.

Questa operazione - la connessione al server MySQL - è da considerare preliminare ad ogni altra operazione sui database. Una volta concluse le operazioni sul database è possibile chiudere la connessione al server in modo esplicito mediante la funzione **mysql_close()** in questo modo:

```
mysql_close($myconn);
```

Selezionare un database

Una volta stabilita la connessione è necessario selezionare uno specifico db sul quale lavorare. A questo scopo PHP ci fornisce la

funzione `mysql_select_db()` da utilizzarsi con la seguente sintassi:

```
mysql_select_db(database,  
connessione);
```

Ad esempio:

```
mysql_select_db('mio_database',  
$myconn) or die('Errore...');
```

Effettuare una query

Per prima cosa vediamo come è possibile recuperare (leggere) dei dati presenti nel nostro database. Per fare questo dobbiamo formulare ed eseguire una *query*, la quale consiste in una interrogazione che lo sviluppatore rivolge al database utilizzando il linguaggio SQL. Per fare ciò si fa ricorso alla funzione `mysql_query()` con

la seguente sintassi:

```
mysql_query(query, connessione);
```

Ad esempio:

```
$query = "SELECT * FROM tabella  
WHERE id > 100";
```

```
$result = mysql_query($query,  
$myconn) or die('Errore...');
```

Si noti che l'indicazione della connessione è facoltativa; se omessa lo script utilizzerà l'ultima connessione aperta.

Leggere i record restituiti da una query di SELECT

Una volta "recuperati" i dati dal database mediante una *SELECT* dovremo preoccuparci di

ciclarli, ad esempio, per stamparli a video. A tal fine ci serviranno una serie di altre funzioni come, ad esempio, **mysql_num_rows()** e **mysql_fe**

Vediamo quindi un esempio completo: poniamo di voler recuperare dalla tabella "amici" una serie di dati (nome, cognome e telefono) e di volerli stampare a video per ogni occorrenza trovata nel nostro database. Ecco il codice completo del nostro script PHP opportunamente commentato:

```
<?php
```

```
// mi connetto al MySQL Server
```

```
$myconn = mysql_connect('localhost',  
'pippo', 'xxxxxx') or die('Errore...');
```

```
// seleziono il database degli amici

mysql_select_db('database_degli_amici',
$myconn) or die('Errore...');

// imposto ed eseguo la query

$query = "SELECT nome, cognome,
telefono FROM amici ORDER BY
cognome ASC";

$result = mysql_query($query,
$myconn) or die('Errore...');

// conto il numero di occorrenze trovate
```

nel db

```
$numrows = mysql_num_rows($result);
```

```
// se il database è vuoto lo stampo a  
video
```

```
if ($numrows == 0){
```

```
    echo "Database vuoto!";
```

```
}
```

```
// se invece trovo delle occorrenze...
```

```
else
```

```
{
```

```
    // avvio un ciclo for che si ripete per il
```


numero di occorrenze trovate

```
for ($x = 0; $x < $numrows; $x++){
```

```
    // recupero il contenuto di ogni record  
    trovato
```

```
    $resrow = mysql_fetch_row($result);
```

```
    $nome = $resrow[0];
```

```
    $cofnome = $resrow[1];
```

```
    $telefono = $resrow[2];
```

```
    // stampo a video il risultato
```

```
    echo "nome: <b>" . $nome . "</b>
```

```
<br/>";  
  
    echo "cognome: <b>" . $cognome . "  
</b><br/>";  
  
    echo "telefono: <b>" . $telefono . "  
</b>";  
  
}  
  
}  
  
// chiudo la connessione  
mysql_close($myconn);  
  
>
```

Quello che abbiamo fatto qui sopra dovrebbe esservi abbastanza chiaro

(ricordate la lezione sui cicli?)... Resta pertanto da chiarire solo il significato di due funzioni specifiche che abbiamo utilizzato:

- **mysql_num_rows()** - Serve per conteggiare il numero di records trovati all'interno del nostro db sulla base di una data query;
- **mysql_fetch_row()** - Recupera il contenuto dei records trovati. Più precisamente restituisce una array contenente i valori di ogni campo riscontrato nel recordset che potremo poi richiamare specificando il relativo indice numerico.

Di seguito, per completezza, alcune considerazioni in merito al codice del nostro esempio.

In alternativa a `mysql_fetch_row()` avremmo potuto utilizzare **`mysql_fetch_assoc()`**, in tal caso i dati dal recordset sarebbero stati recuperabili mediante l'indicazione del nome del campo (invece che dell'indice numerico). Ad esempio:

```
// usando mysql_fetch_rows() abbiamo  
scritto...
```

```
$nome = $resrow[0];
```

```
// ...usando mysql_fetch_assoc()  
avremmo scritto
```

```
$nome = $resrow['nome'];
```

In alternativa a questi due, infine,

avremmo potuto usare `mysql_fetch_array()` che supporta, indistintamente, entrambe le tecniche di chiamata.

Ancora, nel nostro esempio abbiamo ciclato il recordset utilizzando un comune *ciclo for* ma avremmo potuto usare, forse più correttamente, anche *while* in questo modo:

```
// ...  
  
if ($numrows == 0) {  
    echo "Database vuoto!";  
}  
  
else
```

```
{  
  
    while ($resrow =  
mysql_fetch_row($result)) {  
  
        $nome = $resrow[0];  
  
        $cofnome = $resrow[1];  
  
        $telefono = $resrow[2];  
  
  
        // Stampo a video il risultato  
  
        // ...  
  
    }  
  
}
```

```
// ...
```

Cenni di SQL: INSERT INTO, UPDATE e DELETE

Per finire vediamo brevemente come eseguire altre importanti operazioni con i database attraverso i più comuni comandi del linguaggio SQL.

Con INSERT INTO si inseriscono nuovi dati nel db, con UPDATE si aggiornano dei dati già presenti, con DELETE si cancellano dei dati.

Dal punto di vista di PHP queste operazioni non differiscono tra loro, l'unica cosa che cambia è la query che viene eseguita, ma questo discorso attiene

al linguaggio SQL. Facciamo degli esempi:

Per INSERT INTO useremo:

```
mysql_query("INSERT INTO tabella  
VALUES('valore1','valore2','valore3')");
```

Per UPDATE useremo:

```
mysql_query("UPDATE tabella SET  
campo1='valore1', campo2='valore2',  
campo3='valore3' WHERE id = 1");
```

Per DELETE useremo:

```
mysql_query("DELETE FROM tabella  
WHERE id = 1");
```

Attenzione! se nelle query di UPDATE e DELETE non usiamo la clausola "WHERE" verranno aggiornati/eliminati tutti i record del db!

Facciamo un esempio di utilizzo di queste query; vediamo come cancellare con PHP un record dal nostro database MySQL:

```
<?php
// mi connetto al server MySQL

$myconn = mysql_connect('localhost',
'pippo', 'xxxxxx') or die('Errore...');

// mi connetto al database degli amici

mysql_select_db('database', $myconn)
or die('Errore...');
```

```
// imposto ed eseguo la query
$query = "DELETE FROM tabella
WHERE id = 1";

$result = mysql_query($query,
$myconn) or die('Errore...');

// chiudo la connessione

mysql_close($myconn);

?>
```

Per le altre operazioni basterà sostituire la query mantenendo inalterata la struttura del codice PHP.

Aggiornamento: le nuove

funzioni MySQLi e PDO_MySQL

In questa lezione sono state presentate diverse funzioni della famiglia **mysql_*** le quali, tuttavia, sono considerate deprecate nelle versioni più recenti di PHP a favore delle più moderne

funzioni **MySQLi** e **PDO_MySQL**.

Per quanto riguarda la famiglia di funzioni **mysql_***, fortunatamente, il passaggio è piuttosto semplice in quanto (mantenendo lo stile di programmazione procedurale) sarà sufficiente cambiare il prefisso "mysql_" in "mysqli_" per continuare ad utilizzare buona parte dei nostri vecchi script.

Includere file con PHP: `include()` e `require()`

Nella lezione precedente abbiamo accennato a come interagire con un database MySQL attraverso i nostri script PHP. Come abbiamo visto è necessario specificare - affinché la connessione al database funzioni correttamente - le credenziali di accesso al MySQL. Qualora la nostra

applicazione sia composta di più file sarà, ovviamente, necessario aver cura di ripetere le nostre credenziali in ogni singolo script.

Una simile prassi, in realtà, è decisamente sconsigliabile in quanto sarebbe un inutile spreco di tempo ripetere più volte lo stesso codice, ciò aumenterebbe il rischio di errori e renderebbe, tra l'altro, poco agevole ogni operazione di modifica o aggiornamento (si pensi, ad esempio, ad un'applicazione composta da 20 files: qualora doveste modificare i dati di accesso al vostro DB doveste modificare a mano tutti quanti gli script!).

Al fine di far fronte a questa ed altre

necessità analoghe, il linguaggio PHP ci offre alcuni utili **comandi per gestire le inclusioni** di file: questi sono **include** e **require**.

Mediante questi comandi (*include* e *require* non sono delle funzioni!), quindi, potremo evitare le ripetizioni di codice scrivendo le istruzioni una sola volta all'interno di un file che sarà poi incluso all'interno di tutti gli script che necessitano di quel codice.

Per tornare al nostro esempio di partenza potremmo creare un file "dati-mysql.php" con le credenziali di accesso, la connessione al DBMS e la selezione del DB evitando di ripetere sempre lo stesso codice nei file che lo

richiedono:

```
<?php
```

```
$host = "localhost";
```

```
$user = "pippo";
```

```
$pass = "odiotopolino";
```

```
$database = "miodatabase";
```

```
// mi connetto al DBMS
```

```
$myconn = mysql_connect($host, $user,  
$pass) or die('Errore...');
```

```
//Mi connetto al database
```

```
mysql_select_db($database, $myconn)
```

```
or die('Errore...');
```

```
?>
```

All'interno di tutti i singoli file preposti ad operare sul DB sarà sufficiente includere il file appena visto, in questo modo:

```
include "dati-mysql.php";
```

oppure con

```
require "dati-mysql.php";
```

si noti che sia *include* che *require* possono essere utilizzati, indifferentemente, con e senza le parentesi:

```
// corretto
```

```
include("dati-mysql.php");
```



```
// corretto
```

```
include "dati-mysql.php";
```

Differenza tra include e require

I due

comandi *include* e *require* producono il medesimo risultato; l'unica differenza consiste nella gestione di eventuali errori: nel caso il file da includere non si trovasse *include()* genererà un *warning* mentre *require()* un *fata*

error (bloccando, di fatto, l'esecuzione dello script).

Si noti che, affinché l'inclusione vada a buon fine, è necessario specificare il percorso corretto del file che si desidera includere (nel nostro esempio il file "dati-mysql.php" si trova nella stessa cartella degli script che lo includono).

Per approfondire l'argomento vi consiglio di leggere queste semplici referenze:

- [include\(\)](#)
- [require\(\)](#)

Usare `include_once` e `require_once`

E' da segnalare l'esistenza di due

interessanti varianti, si tratta dei comandi **include_once** e **require_once**. La loro funzione è identica a *include* e *require* con l'unica differenza che prima di includere il file verificano che questo non sia già stato precedentemente incluso nella pagina ed, in tal caso, non fanno nulla.

FINE.

In questa guida ho cercato di trasmettere a voi utenti principalmente le basi del php.

Una volta imparati i concetti chiave è il programmatore che, seguendo una logica deve riuscire a mettere il tutto insieme sviluppando delle applicazioni (O come si chiamano nello specifico per il web: scripts).

Vi linko alcuni siti dove potete trovare scripts pronti, anche da poter modificare o semplicemente "osservare" il codice:

<http://php.html.it/script>

*Spero che questo libriccino ti
sia stato utile!*

**GRAZIE PER
AVER LETTO,**

A handwritten signature in black ink, appearing to read 'Younes Haoufadi', with a long vertical stroke extending downwards from the start of the name.

Younes Haoufadi

All rights are the property of Hacker
Italia Srl and their respective owners

any unauthorized use will be treated according to law.

Tutti i diritti sono di proprietà di Hacker Italia Srl e dei rispettivi titolari ogni utilizzo non autorizzato sarà trattato a norma di legge

**©2018 All rights Reserved
by Hacker Italia Editons
Srl;**

Official Suppliers:

Amazon Media Inc.
StreetLib Srl.
Mondadori spa
Ibs.it
Google Books Inc.
Kobo Inc.
All StreetLib suppliers

Euti7748beyy4v48wb2m