

UNIT 4 – 5 – 6 – 7 - 8

1.1 Gestione delle Sessioni con PHP

Prima di tutto cosa sono le sessioni e a cosa servono?

Le sessioni servono, a prescindere dal linguaggio di programmazione, per abbattere quel limite dell'assenza di stato del protocollo HTTP.

Ovvero il server web una volta ricevuta la richiesta dal client e servita la pagina, le richieste successive che arrivano dallo stesso browser sono considerate da Apache come richieste differenti e completamente slegate alla precedente.

Nasce quindi l'esigenza di avere uno strumento a disposizione del programmatore per poter mantenere informazioni legate per esempio alla login dell'utente, dati forniti etc...

Una sessione è gestita da PHP tramite l'identificativo sull'hash MD5 dell'indirizzo IP remoto, dell'ora corrente e di altri dati casuali.

Tale dato può essere registrato sia sotto forma di cookie, sia appeso all'indirizzo URL come ?PHPSESSID=xxx manualmente o in modo automatico abilitando l'opzione session.use_trans_sid in php.ini.

Per motivi di sicurezza è vivamente consigliato forzare il client ad attivare i cookie in quanto vi è una maggiore sicurezza onde evitare che qualcuno possa individuare il codice identificativo di una sessione e utilizzare i privilegi e i dati presenti in quella sessione.

Esempio pratico per forzare utilizzo dei cookie sulle sessioni:

```
<?php  
  
ini_set("session.use_cookies", 1);  
ini_set("session.use_only_cookies", 1);  
  
?>
```

In questo modo con la prima riga di codice si indica allo script che il dato relativo alla sessione sarà passato per cookie, con la seconda riga invece si forza PHP ad accettare solo identificativo di sessione che arrivino via cookie e ad ignorare quelli derivanti da URL.

Riferimenti funzioni di settaggio valori di configurazione in PHP

<http://it.php.net/manual/it/function.ini-set.php>

<http://it.php.net/manual/it/ini.php#ini.list>

Codice essenziale utilizzo sessioni

Vediamo un esempio di codice con commenti per far comprendere il funzionamento delle sessioni e relativi riferimenti al manuale di php.net

```
<?php

/*
Avvio della sessione all'interno di questo script
http://it.php.net/manual/it/function.session-start.php
*/
session_start();

/*
Settaggio del valore derivante da un ipotetico modulo HTML mediante POST del nome
del navigatore, in questo modo potremo memorizzare il suo nome per tutta la sua
sessione di navigazione.
*/
$_SESSION['name'] = $_POST['name'];

/*
Distruzione della sessione corrente ed equivalente a chiusura del browser
http://it.php.net/manual/it/function.session-destroy.php
*/
session_destroy();

?>
```

Per approfondire il funzionamento delle sessioni in PHP, alcuni aspetti sistemistici e sempre per prepararsi alla certificazione Zend visitare e studiare la parte di manuale online di php.net relativa alle sessioni:

<http://it.php.net/manual/it/ref.session.php>

Approfondire non solo gli aspetti di programmazione pura con le sessioni ma anche gli aspetti di sicurezza e sistemistici.

1.2 Array numerici e associativi

Gli array in PHP come abbiamo già avuto modo di dire sono identificati da un legame chiave ==> valore.

Tale chiave in PHP può essere sia un riferimento numerico, sia un riferimento stringa.

In pratica nel caso di riferimento numerico la chiave assume un valore da 0 a x. Va però detto che in PHP è assolutamente possibile avere un array di tipo misto, ovvero un array che ha chiavi di tipo numerico che di tipo associativo.

Vediamo ora due esempi:

```
$arr_num = array(1, 2, 3);  
print_r (arr_num);
```

```
// OUTPUT  
Array  
(  
    [0] => 1  
    [1] => 2  
    [2] => 3  
)
```

In sostanza all'elemento chiave 0 corrisponde valore 1, alla chiave 1 il valore 2 e via discorrendo.

Vediamo ora l'esempio associativo:

```
$arr_ass = array("nome" => "Massimo", "cognome" => "Caselli", "Eta'" => 26);  
print_r (arr_ass);
```

```
// OUTPUT  
Array  
(  
    [nome] => Massimo  
    [cognome] => Caselli  
    [Eta'] => 26  
)
```

Come possiamo notare l'array di tipo associativo è più lungo e complesso da impostare ma offre un riferimento chiave molto semplice da ricordare essendo di tipo stringa e il cui significato può essere legato a ciò che il suo valore esprime.

L'array associativo viene fortemente utilizzato per il recupero dei dati da query eseguite su MySQL o su qualche DB server.

Per approfondimenti:

<http://it.php.net/manual/it/function.array.php>

UNIT 3 A

1.1 Gestione avanzata e manipolazione di array in PHP

Tra le funzionalità più evolute di PHP ci sono sicuramente quelle legate alla manipolazione di array monodimensionali e multidimensionali.

Vediamo di seguito come manipolare in modo semplice gli array.

Innanzitutto per creare una struttura dati di questo tipo si utilizza `array()`

ad esempio per creare una piccola rubrica `$rubrica` possiamo procedere in questo modo:

```
$rubrica = array (  
    "gino" => "0123456789",  
    "lino" => "987654321",  
    "tino" => "135792468");
```

Si noti l'utilizzo della sintassi **indice => valore** per associare ad un particolare indice (numerico o stringa) un particolare valore.

Nel caso venga omissa l'indice allora i valori saranno indicizzati tramite un valore numerico incrementale, si noti che in tale caso l'indice partirà da 0

Molto utile per verificare il valore degli array (e non solo) è la funzione `print_r ($variabile)` che restituisce in output una versione "leggibile" dell'array

Per ottenere un nuovo array con solamente gli indici dell'array iniziale:

```
array_keys($array)
```

tornando al nostro esempio l'istruzione

```
print_r (array_keys($rubrica))
```

avrà come risultato

```
Array  
(  
[0] => gino  
[1] => lino  
[2] => tino  
)
```

In modo analogo si utilizza

```
array_values($array)
```

per avere un nuovo array in cui ci saranno gli stessi valori dell'originale ma indicizzati

numericamente.

Quindi

```
print_r (array_values($rubrica))
ci mostrerà:
Array
(
[0] => 0123456789
[1] => 987654321
[2] => 135792468
)
```

Nel caso ci serva solo verificare che il particolare indice sia presente nell'array, invece che scorrerlo per intero, si può utilizzare la funzione

```
array_key_exists($indice, $array)
```

che ci restituirà TRUE nel caso in cui \$indice sia presente come chiave in \$array.

Per effettuare operazioni su ogni valore dell'array possiamo usare un semplice ciclo while associato alla funzione `each($array)`

Ad esempio:

```
while (list ($nome, $telefono) = each ($rubrica))
{
print "$nome ha telefono $telefono
";
}
}
```

ci mostrerà l'elenco completo presente nella nostra rubrica. Da notare come, prima di effettuare una eventuale nuova lettura dell'array sia necessario riportare il puntatore al primo indice del nostro array, questo avviene utilizzando la funzione:

```
reset()
```

che restituisce anche il primo valore dell'array.

Le classiche funzioni POP e PUSH utilizzabili nella gestione di array come stack, cioè come una coda LIFO, sono

```
array_pop($array)
```

che estrae e restituisce l'ultimo valore di \$array, riducendone quindi la lunghezza e

```
array_push($array, $elemento)
```

che aumenta la lunghezza di \$array mettendovi in cima \$elemento

Questo può essere verificato contando gli elementi presenti nel nostro \$array, facciamolo con

```
count($array) oppure sizeof($array)
```

che ritornano il numero di elementi presenti nell'array oppure 1 se la variabile passata

non è un array oppure 0 nel caso di
`count(NULL)`

Questa veloce panoramica si chiude con le funzioni dedicate all'ordinamento degli array:

`sort($array, $flag)`

che ridispone gli elementi dell'array dal più piccolo al più grande, questo avviene ridefinendo gli indici, attenzione quindi al fatto che gli indici originali verranno persi, tornando al nostro esempio

```
sort($rubrica);  
print_r ($rubrica);
```

avrà come risultato;

```
Array  
(  
    [0] => 0123456789  
    [1] => 135792468  
    [2] => 987654321  
)
```

è possibile definire la modalità di comparazione fra gli elementi tramite l'impostazione di `$flag`

`sort($array, SORT_REGULAR): default`, vengono comparati in modo normale
`sort($array, SORT_NUMERIC): default`, vengono comparati in modo numerico
`sort($array, SORT_STRING): default`, vengono prima convertiti in stringhe e poi comparati

`rsort($array, $flag)`

reverse sort si comporta come sort solo che gli elementi verranno disposti in ordine inverso

`asort($array, $flag)` e `arsort($array, $flag)`

sono le funzioni di ordinamento che permettono di mantenere l'associazione del valore con l'indice originale

```
asort($rubrica);  
print_r ($rubrica);
```

avrà come risultato;

```
Array  
(  
    [gino] => 0123456789  
)
```

[tino] => 135792468

[lino] => 987654321

)

QUESTA PARTE E' ESSENZIALE PER LA CERTIFICAZIONE ZEND IN QUANTO E' UNO DEI CAPITOLI CHE PRESENTA PIU' DOMANDE ALL'ESAME, INOLTRE E' ANCHE MOLTO IMPORTANTE PER IL NORMALE SVILUPPO DI APPLICAZIONI WEB.

Tutte le informazioni si trovano qua:

<http://it.php.net/manual/it/ref.array.php>

1.1 Formattazione e manipolazione stringhe in PHP

Una delle parti decisamente più consistenti di PHP è quella rappresentata dalla gestione e manipolazione delle stringhe. Tale parte è inoltre molto richiesta e importante per ottenere la certificazione Zend.

Sul manuale online di PHP è disponibile tutta la gamma di funzioni:

<http://it.php.net/manual/it/ref.strings.php>

FUNZIONI ECHO() e PRINT()

Di seguito andremo ad analizzare alcune funzioni affrontando man mano dei casi i più reali possibili.

Definiamo ad esempio la stringa:

```
$a = 'questo è il testo della variabile $a';
```

Notiamo subito l'utilizzo di ' anziché di " in quanto abbiamo la necessità che \$a sia stampata come stringa e non con il contenuto della relativa variabile \$a.

La funzione di stampa più comune che abbiamo visto è:

```
echo $a;
```

La funziona echo(), in realtà non è una funzione ma un costrutto del linguaggio e quindi non richiede l'utilizzo di parametri.

Esiste una modalità abbreviata per utilizzare echo, ovvero:

```
<?=$a?>
```

che equivale a:

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

Il tutto espresso con una sola istruzione di apertura e chiusura dei tag PHP. Da notare che l'utilizzo di tale modalità richiede l'attivazione della configurazione short_open_tag attiva.

Il medesimo funzionamento di echo() esiste anche per la funzione print().

Per la differenza tra le due visionare il link seguente:

http://www.faqs.com/knowledge_base/view.phtml/aid/1/fid/40

FUNZIONE PRINTF() e SPRINTF()

Una più potente funzione di stampa di variabili e stringhe è rappresentata da printf()

```
printf($formato, $argomenti), sprintf($formato, $argomenti)
```

Esse permettono di inserire all'interno della stringa \$formato dei posizionatori tipizzati che verranno sostituiti dagli argomenti presenti nella chiamata alla funzione.

Ad esempio:

```
printf("Gentile cliente %s hai %u prodotti nel carrello", $cognome, $num_prod);
```

Produrrà qualcosa del tipo:

```
Gentile cliente Rossi hai 3 prodotti nel carrello"
```

La funzione `sprintf()` si differenzia solamente per il fatto che essa ritorna una stringa invece che stamparla direttamente, il codice seguente ha lo stesso effetto dell'esempio precedente:

```
$stringa = sprintf("Gentile cliente %s hai %u prodotti nel carrello", $cognome, $num_prod);  
echo $stringa;
```

FUNZIONE CHR()

La funzione `chr()` restituisce il corrispondente carattere ascii passato come argomento alla stessa funzione.

Ovvero ad esempio:

```
echo chr(37);
```

Restituisce il carattere associato: %

Per la lista completa dei caratteri ASCII: <http://www.lookuptables.com/>

FUNZIONI DI TRIM(), LTRIM(), RTRIM()

Talune volte capita che l'utente quando inserisce il dato per sbaglio o per qualche copia e incolla non perfetto inserisca a sinistra o a destra di una stringa uno o più caratteri blank.

Per ovviare a tale problema esistono le funzioni sopra citate.

Vediamo un esempio:

```
$a = ' questa è la mia stringa '
```

Con l'utilizzo della funzione `trim($a)` otterremo questo output:

questa è la mia stringa (senza spazi né a destra né a sinistra)

Con `rtrim($a)` otterremo:

questa è la mia stringa (con spazio a sinistra)

Ovviamente con `ltrim($a)` otterremo l'equivalente di `rtrim()` solo che lo spazio che sarà eliminato sarà quello di sinistra anziché quello di destra.

QUESTE FUNZIONI SONO MOLTO UTILI IN FASE DI CHECK DI LOGIN E PASSWORD

PER EVITARE ERRORI DI LOGIN DOVUTI A DIGITAZIONI SBAGLIATE.

ALTRE FUNZIONI DI MANIPOLAZIONE STRINGHE

Una ulteriore accortezza è la codifica/decodifica dei caratteri speciali in entità HTML. Per entità HTML si intendono quelle sequenze di caratteri preceduti da & che vengono utilizzati per indicare ai browser di visualizzare il carattere speciale, alcuni esempi sono:

"
&
€

La funzione:

`htmlentities($stringa)`

Converte automaticamente tutti i caratteri per i quali esiste una codifica HTML, mentre:

`htmlspecialchars($stringa)`

Converte solamente i caratteri &, ", ', < e >

La conversione opposta viene effettuata dalla funzione:

`html_entity_decode($stringa)`

Passiamo ora ad analizzare le funzioni che ci permettono di operare sul contenuto della stringa per estrarne dei sottoinsiemi di caratteri (sotto-stringhe) o per verificare che all'interno della stringa appaiano particolari caratteri. Queste funzioni sono spesso usate in costrutti condizionali per effettuare operazioni solo sulle stringhe che soddisfano particolari condizioni.

`substr($stringa, $inizio, $lunghezza)`

Questa funzione ci permette, specificando opportunamente i parametri `$inizio` e `$lunghezza` (quest'ultimo non obbligatorio), di estrapolare dalla stringa iniziale delle sequenze di caratteri, la cosa interessante è che oltre all'utilizzo intuitivo:

```
$stringa = "abcdefghijklmnpqrstuvz";
```

```
print substr($stringa, 5);
```

Che produrrà:

```
fgjilmnpqrstuvz
```

Restituendo i caratteri a partire dal sesto fino al termine della stringa, oppure:

```
print substr($stringa, 0, 10);
```

Che produrrà:

abcdefghijkl

Estraendo i primi 10 caratteri a partire dal primo (si noti come, così come per gli array il primo carattere ha indice 0).

E' anche possibile inserire numeri negativi come parametri, infatti nel caso l'indice di partenza sia negativo i caratteri verranno presi a partire dal fondo della stringa, ad esempio:

```
print substr($stringa, -5);
```

produce l'output:

stuvz

Tornando cioè gli ultimi cinque caratteri.

Nel caso sia negativo il parametro che indica la lunghezza il numero di caratteri ritornati sarà uguale al numero di caratteri da \$inizio alla fine della stringa meno il valore negativo di \$lunghezza, vediamo degli esempi:

```
print substr($stringa, 0, -5);
```

Torna:

abcdefghijklmno

Cioè i caratteri dalla prima posizione fino al sestultimo, mentre

```
print substr($stringa, 5, -5);
```

Torna:

ghijklmno

quindi dal sesto al sestultimo,

```
print substr($stringa, -5, -3);
```

Produce:

st

cioè a partire dal quint'ultimo carattere prendo i caratteri fino al terz'ultimo, si noti come (ovviamente) chiamate del tipo:

```
print substr($stringa, -n, -n);
```

Dove n è un intero producano sempre una stringa vuota.

Una funzione basata sugli stessi concetti di substr() è:

```
substr_replace($stringa, $nuova, $inizio, $lunghezza);
```

Che inserisce al posto della sottostringa individuata da \$inizio e \$lunghezza la stringa \$nuovo, così:

```
print substr_replace($stringa, 'STUVZ', -5);
```

produce l'output:

```
abcdefghijklmnpqrSTUVZ
```

e invece:

```
print substr_replace($stringa, 'ST', -5, -3);
```

produce l'output:

```
abcdefghijklmnpqrSTuvz
```

Un'altra possibilità per la ridefinizione di una stringa è data da:

```
str_replace($trova, $sostituisci, $stringa);
```

Che sostituisce all'interno di \$stringa tutte le occorrenze di \$trova con \$sostituisci, ad esempio:

```
$stringa="alfabeto alfanumerico";
```

```
print str_replace("alfa", "ALFA", $stringa);
```

ci mostra

```
ALFAbeto ALFAnumerico
```

Attenzione che il match della sottostringa sarà case-sensitive cioè:

```
$stringa="Alfabeto alfanumerico";
```

```
print str_replace("alfa", "ALFA", $stringa);
```

Produrrà:

```
Alfabeto ALFAnumerico
```

Solo dalla versione 5 di PHP è stata introdotta la funzione:

```
stri_replace($trova, $sostituisci, $stringa)
```

Che effettua un math case insensitive.

```
strpos($stringa, $cerca);
```

ritorna la posizione all'interno di \$stringa del carattere o sottostringa \$cerca oppure FALSE se la sottostringa non è presente:

```
$stringa="Alfabeto alfanumerico";  
print strpos($stringa, "fa");
```

Ci mostra:

2

Indicandoci che la prima occorrenza di "fa" parte dal terzo carattere di \$stringa. Un problema comune si ha quando si utilizza questa funzione per verificare la presenza o meno della sottostringa, infatti impostando il costrutto condizionale in questo modo

```
if (!strpos($stringa, "AI"))  
print "AI (sembra) non esserci...";
```

Incappiamo nel fatto che il PHP valuta la condizione !strpos(\$stringa, "AI") come FALSE dato che essa vale 0 essendo che la stringa AI parte dall'indice 0 della stringa... Per ovviare a questo problema si può utilizzare l'operatore di confronto === che indica un'uguaglianza non solo nel valore ma anche nel tipo delle due variabili confrontate, il costrutto "giusto" sarà del tipo:

```
if (strpos($stringa, "AI")==FALSE)  
print "AI non c'è";  
else  
print "AI c'è";
```

Se ci interessa trovare invece l'indice dell'ultima occorrenza della sottostringa possiamo usare la funzione

right strpos:

```
strrpos($stringa, $cerca)  
utilizzando l'esempio precedente  
$stringa="Alfabeto alfanumerico";  
print strrpos($stringa, "fa");  
vediamo come il risultato sia diverso  
11
```

Abbiamo visto come le stringhe in PHP siano molto vicine al concetto di array, è infatti possibile immaginare una stringa come un array di caratteri indicizzati a partire da 0, possiamo infatti accedere al singolo carattere con modalità del tipo

```
$stringa = "abcdefghijklmnopqrstuvz";  
print $stringa{12};
```

che stampa il tredicesimo carattere di `$stringa`:

0

Questa "vicinanza" tra stringhe ed array può essere ancora meglio compresa analizzando le funzioni che si occupano proprio di permettere il passaggio tra una struttura "stringa" ad un "array" e l'inverso.

Partiamo da:

```
explode($separatore, $stringa)
```

che ritorna un array con, in ogni elemento, la sottostringa ottenuta suddividendo `$stringa` in base al carattere o sottostringa `$separatore`.

Vediamo un esempio

```
$stringa = "Mario Rossi studente";  
$array = explode(" ", $stringa);  
print "Nome: ".$array[0]."\n<br>";  
print "Cognome: ".$array[1]."\n<br>";  
print "Occupazione: ".$array[2]."\n<br>";
```

Produce come output

Nome: Mario

Cognome: Rossi

Occupazione: studente

Un utilizzo classico di questa funzione è il parsing di file di tipo CSV (Comma Separated Value) cioè di file che contengono un record per riga, i campi del record sono separati da un carattere predefinito.

Si presti attenzione ai casi particolari in cui

- `$separatore` non sia presente all'interno di `$stringa`: il risultato della funzione sarà un array che contiene `$stringa`

- `$separatore` è una stringa vuota, in tal caso `explode()` ritorna il valore FALSE

L'operazione inversa, cioè partire da un array per ottenere una stringa è possibile con `implode($separatore, $array)`

che genera la stringa unendo i vari elementi di `$array` con `$separatore`, così

```
$array = array("nome" => "Mario", "cognome" => "Rossi", "occupazione" => "studente");
```

```
$stringa = implode(", ", $array);
```

```
print $stringa;
```

genera un riga che potremmo inserire in un file CSV:

Mario,Rossi,studente

E' anche possibile effettuare un suddivisione della stringa basandosi su un espressione regolare, la funzione che ce lo permette è

```
split($espressione, $stringa)
```

Questo può essere utile ad esempio quando la stringa sia inserita da un utente che può quindi utilizzare diversi separatori convenzionali, classico esempio è quello di reperimento del prefisso e del numero di telefono ad esempio:

```
$telefono1 = "335 12345678";  
$telefono2 = "335-12345678";  
$array1 = split ("[-]", $telefono1);  
$array2 = split ("[-]", $telefono2);  
print_r ($array1);  
print_r ($array2);
```

Si noti come l'utilizzo di `split()` sia consigliato solo quando sia effettivamente necessario utilizzare espressioni regolari.

Per maggiori informazioni e per la preparazione alla certificazione Zend consultare e studiare il manuale di PHP online relativamente alle stringhe:

<http://it.php.net/manual/it/index.php>

1.1 Utilizzo dei files con PHP

Il linguaggio di programmazione PHP ha una consistente parte relativa all'utilizzo dei files, l'upload da moduli HTML e diverse decine di funzioni utili per lavorare con i files in PHP.

Come sempre la nostra bibbia anche in considerazione della certificazione Zend è rappresentata dal manuale online di php.net nella sezione relativa al filesystem:

<http://it2.php.net/manual/it/ref.filesystem.php>

Vediamo di seguito passo un semplice utilizzo delle funzionalità relative ai files in PHP

Apertura file con PHP

Per aprire un files utilizziamo fopen()

Tabella 1. Elenco dei possibili valori usati da fopen() per il parametro *mode*

<i>mode</i>	Descrizione
'r'	Aprire in sola lettura; posiziona il puntatore all'inizio del file.
'r+'	Aprire in lettura e scrittura; posiziona il puntatore all'inizio del file.
'w'	Aprire il file in sola scrittura; posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
'w+'	Aprire in lettura e scrittura; posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
'a'	Aprire in sola scrittura; posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.
'a+'	Aprire in lettura e scrittura; posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.
'x'	Crea ed apre il file in sola scrittura; posiziona il puntatore all'inizio del file. Se il file esiste già la chiamata a fopen() fallirà restituendo FALSE e verrà generato un errore di livello E_WARNING . Se il file non esiste si tenterà di crearlo. Questo equivale a specificare i flag O_EXCL O_CREAT nella sottostante chiamata a <i>open(2)</i> . Questa opzione è supportata a partire dalla versione 4.3.2 di PHP, e funziona solo con i file locali.
'x+'	Crea ed apre il file in lettura e scrittura; posiziona il puntatore all'inizio del file. Se il file esiste già la chiamata a fopen() fallirà restituendo FALSE e verrà generato un errore di livello E_WARNING . Se il file non esiste si tenterà di crearlo. Questo equivale a specificare i flag O_EXCL O_CREAT nella sottostante chiamata a <i>open(2)</i> . Questa opzione è supportata a partire dalla versione 4.3.2 di PHP, e funziona solo con i file locali.

Tipicamente l'utilizzo è:

```
$miarisorsafile = fopen("pathtofile/miofile.txt", "r+");
```

In questo modo ad esempio apriamo il file in modalità di lettura/scrittura e

posizioniamo il puntatore all'inizio del file.

Scrittura di un file con PHP

```
fwrite( $miarisorsafile, $stringa );
```

Questa funzione scrive un file salvaguardando la corrispondenza binaria. In pratica viene scritto tutto il contenuto di \$stringa all'interno del file PHP.

A questo punto basta esclusivamente chiudere la risorsa al file con `fclose($miarisorsafile);`

1.2 Upload di files con PHP

Una delle operazioni più comuni che si fanno con PHP è quella di permettere il caricamento di files via web.

Innanzitutto è necessario che sia abilitato il supporto in `php.ini`

Di seguito vediamo il codice commentato di un modulo per il caricamento di files e del suo relativo codice:

```
<html>
<head></head>
<body>

<form action="/upload.php" method="post" enctype="multipart/form-data">
<br><br>
Choose a file to upload:<br>
<input type="file" name="file"><br>
<input type="submit" name="submit" value="submit">
</form>

</body>
</html>
```

Questo è un semplice modulo che permette all'utente di caricare un proprio file locale sul server web.

Vediamo ora il codice della pagina `upload.php` che si occuperà del vero e proprio caricamento del file.

Prima di tutto individuiamo in `upload.php` quali variabili relative al file avremo:

```
$_FILES['file'] "file" E' il nome file inserito nel form
$_FILES['file']['name'] Il nome originale del file dal computer dell'utente
$_FILES['file']['tmp_name'] Nome temporaneo del file quando viene caricato

$_FILES['file']['type'] Il nome del file generato dal browser dell'utente

$_FILES['file']['size'] La dimensione del file, in byte
```

Controlliamo ora che il file sia stato caricato correttamente:

```
if (!is_uploaded_file($_FILES['file']['tmp_name']))
{
    $error = "Upload fallito";
    // Elimino il file temporaneo mal caricato
    unlink($_FILES['file']['tmp_name']);
}
else
{
    // File caricato correttamente e spostamento del file nella destinazione corretta
    move_uploaded_file ( $_FILES['tmp_name'], "/pathtonewfile/file.ext" );
}
```

E' suggerito caldamente anche dalla Zend di utilizzare la funzione `move_uploaded_file()` anziché `copy` per questioni relative all'ottimizzazione dello spazio disco del server visto che il file temporaneo viene spostato e non copiato, in questo modo si evitano sprechi di spazio disco.

Ovviamente comunque PHP ripulisce ogni tot tempo i files temporanei caricati.

Naturalmente è possibile archiviare files su filesystem e memorizzare la loro location su DB, filtrare per tipologia di files etc...

Per maggiori informazioni e per la certificazione Zend:

<http://it.php.net/manual/it/ref.filesystem.php>

1.1 Gestione delle date in PHP

Il linguaggio di programmazione PHP ha moltissime funzioni per la gestione delle date di cui moltissime basate su **UNIX timestamp**.

Lo UNIX timestamp equivale al numero di secondi trascorsi dal 1° gennaio 1970 alle 00:00:00 GMT.

Vediamo di seguito le funzioni principali delle date di PHP

Funzione `time()`

Restituisce il numero di secondi dal 1° gennaio 1970. La UNIX Epoch.

Ad esempio:

```
echo time(); // Restituisce qualcosa tipo 1077811877
```

Funzione che formatta la data e l'ora.

Per maggiori informazioni:

<http://it.php.net/manual/it/function.time.php>

Funzione `date("[argomenti]")`

ARGOMENTI (i più usati):

a stampa "am" o "pm"

A stampa "AM" o "PM"

d stampa il giorno del mese in formato numerico in 2 cifre (da "01" a "31")

D stampa il giorno della settimana in formato testuale abbreviato a 3 lettere (es. "Mon")

F stampa il mese in formato testuale (es. "January")

h stampa l'ora nel formato a 12-ore (es. da "01" a "12")

H stampa l'ora nel formato a 24-ore (es. da "00" a "23")

i stampa i minuti (es. da "00" a "59")

I (I maiuscola) stampa "1" se c'è l'ora legale, "0" se c'è quella solare

l (L minuscola) stampa il giorno della settimana in formato testuale non abbreviato (es. "Monday")

L stampa "1" se l'anno è bisestile, "0" altrimenti

m stampa il mese in formato numerico (es. da "01" a "12")

M stampa il mese in formato testuale abbreviato a 3 lettere (es. "Jan")

s stampa i secondi (es. da "00" a "59")

Y stampa l'anno nel formato a 4 cifre (es. "2002")

y stampa l'anno nel formato a 2 cifre (es. "02")

Es. `echo date ("l F Y h:i:s A");` darà come risultato Thursday October 2002 3:35:40 PM

Per prepararsi alla certificazione Zend e approfondire `date()`:

<http://it.php.net/manual/it/function.date.php>

Funzione `mktime()`

Restituisce la UNIX timestamp per una data. Questa timestamp è un intero lungo (longint) contenente il numero di secondi tra la Unix Epoch (ossia "January 1 1970") e la data e orario specificati. E' usata per fare calcoli tra date.

Viene usato in combinazione con `date`

`mktime (int hour, int minute, int second, int month, int day, int year)`

Vediamo un esempio:

```
echo date ("M-d-Y", mktime (0,0,0,8,10,02));
```

con il comando `mktime (0,0,0,8,10,02)` viene indicata la data da esaminare (le 00:00:00 del 10 ottobre 2002) e con il comando `echo date ("M-d-Y")` viene detto come trattare la data e l'ora appena calcolata.

Alla fine verrà stampata la stringa Oct-10-2002.

L'ultimo giorno del mese viene espresso come il giorno "0" del mese successivo.

```
Ad esempio: echo date ("d M Y", mktime (0,0,0,10,0,02));
```

Indica l'ultimo giorno del mese di Settembre 2002. Verrà quindi stampato 30 Sept 2002.

Per `mktime()` documentazione ufficiale:

<http://it.php.net/manual/it/function.mktime.php>

Funzione `checkdate(int mese, int giorno, int anno)`

Verifica una data standard passando i tre argomenti di mese, giorno e anno.

La data è ritenuta valida secondo questa tabella:

- Anno è compreso tra 1 e 32767
- Mese è compreso tra 1 e 12
- Giorno è compreso tra il numero dei giorni possibile per il mese dato. Gli anno(i) bisestili sono presi in considerazione

Giorno è compreso tra il numero dei giorni possibile per il mese dato. Gli anno(i) bisestili sono presi in considerazione.