

PHP: Introduzione

- ❑ Nato come linguaggio dinamico per scrivere pagine web personali
 - ❖ Nato come **P**ersonal **H**ome **P**age
 - ❖ Ora: **P**HP: **H**ypertext **P**reprocessor
- ❑ È alla versione 5 e contempla una serie grandissima di librerie per
 - ❖ gestione stringhe
 - ❖ accesso a vari database (MySQL, PostGres, etc)
 - ❖ XML
 - ❖ cookies
 - ❖ creazione dinamica di immagini
 - ❖ ...
- ❑ Open source, per ogni piattaforma
- ❑ Current version: 5.x

PHP: documentazione

- ❑ Sito ufficiale: <http://www.php.net>
 - ❖ sorgenti, documentazione online anche in italiano
- ❑ Altri siti utili:
 - ❖ <http://www.phpbuilder.com> (community con articoli specifici e forum)
 - ❖ <http://www.hotscripts.com> (scripting in generale, esempi e programmi pronti)

PHP: caratteristiche principali

- ❑ Linguaggio interpretato
- ❑ Sintassi di derivazione C
- ❑ Utilizzabile sia in modo procedurale tradizionale che object-oriented (con limitazioni)
- ❑ Linguaggio «loosely typed»
 - ❖ Non è necessario dichiarare le variabili
 - ❖ Conversione automatica tra tipi quando necessario

Incorporare PHP nelle pagine HTML

- ❑ I files PHP debbono essere inseriti nella parte di file system visibile al server web, insieme alle normali pagine HTML
- ❑ Nel caso di XAMPP, le pagine sono da porsi nella cartella **htdocs**
- ❑ Solitamente le pagine HTML che incorporano codice PHP hanno estensione **.php**
 - ❖ Web servers che supportano il PHP passano queste pagine automaticamente all'interprete PHP
 - ❖ Comunque, comportamento configurabile (es. passare tutti i .html per non mostrare che php è usato)

Incorporare PHP nelle pagine HTML

- ❑ Le pagine HTML pure non vengono modificate dall'interprete PHP
- ❑ Il codice PHP può essere incorporato nelle pagine in differenti modi:
 - ❖ Dentro a una «php preprocessing instruction»:
`<?php ... ?>` (**RACCOMANDATO!**) o tramite tag di «general processing» (se configurati in php.ini): `<? ... ?>` `<% ... %>`
 - ❖ All'interno di HTML script elements:
`<script language="php">.....</script>`

Iniziare con il PHP

- Un metodo semplice per verificare che l'interprete PHP sia configurato e operi correttamente è chiamare la funzione **phpinfo()**
 - ❖ Accedere tramite un browser al server web all'URL che corrisponde ad una pagina il cui contenuto è:

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

Sintassi di base

- ❑ Fine statement: ; (punto e virgola) - obbligatorio, si possono anche spezzare gli statement su più linee.
- ❑ Commenti:
monolinea: `//commento`
`#altro commento`
multilinea `/* righe di commenti ... */`
- ❑ Le keywords, come le funzioni e i metodi, sono case INsensitive
- ❑ Il resto è case sensitive (es. variabili)

Tipi di dati

□ Tipi scalari:

- ❖ bool o boolean (valori TRUE e FALSE)
- ❖ Integer (dipendente dal sistema, tipicamente 32-bit, sintassi C per le costanti)
- ❖ float (dipendente dalla piattaforma, tipicamente doppia precisione, sintassi C per le costanti)
- ❖ string

□ Tipi composti:

- ❖ array (indicizzati e associativi)
- ❖ oggetti

□ Per verificare il tipo si usano le funzioni

`is_int()`, `is_float()`, `is_bool()`,
`is_string()`

Stringhe

□ Le stringhe possono essere scritte usando varie notazioni (possono coprire più linee):

❖ Singoli apici

- Non viene fatta l'espansione delle sequenze di escape e delle variabili, tranne `\'` per l'apice singolo e `\\` per il backslash
- Esempio: `'Sample string with \'' and '\\ '`

❖ Doppi apici

- Espansione delle sequenze di escape e delle variabili
- Esempio:

```
$color = 'white';  
echo "the horse\nis $color"
```

Stampa: the horse
 is white

Costanti

- ❑ È possibile associare dei nomi a delle costanti

```
define('Nomecostante', Valore)
```

```
define("color", "rosso");
```

```
define('prezzo', 13.50);
```

```
echo color; //stampa rosso
```

Magic constants

- ❑ Ci sono delle costanti simboliche predefinite, che iniziano per doppio underscore (__)

<code>__FILE__</code>	the full name of the current file
<code>__DIR__</code>	the directory of the current file
<code>__LINE__</code>	the current line number
<code>__FUNCTION__</code>	the current function name
<code>__CLASS__</code>	the current class name
<code>__METHOD__</code>	the current method name
<code>__NAMESPACE__</code>	the current namespace

Operatori utili

- ❑ Operazioni aritmetiche: `+` `-` `*` `/` `++` `--`
 - ❖ (non c'è la divisione intera! (cioè con troncamento), il risultato è float a meno che il resto sia zero)
- ❑ Operazioni logiche: `==`, `===`, `!=`, `!==`, `!`, `NOT`, `&&`, `AND`, `||`, `OR`,
- ❑ Concatenazione di stringhe: `.` (punto)
 - ❖ es. `$s = "Lato " . "server" ;`
 - ❖ abbreviato: `$s .= "stringa"` equivale a `$s=$s."stringa«`
- ❑ Prestare attenzione alla precedenza degli operatori

Variabili

- ❑ In PHP, le variabili sono prefissate da **\$** (segno di dollaro)
- ❑ Le variabili iniziano con una lettera o _ e contengono solo lettere, cifre o underscore (_)
- ❑ **Sono case sensitive**
- ❑ Le variabili possono essere locali o globali
- ❑ Esempi:

```
$a = "prova";
```

```
$b = 18;
```

```
$c = $b+77;
```

Esempi

□ L'operatore di assegnamento è l'uguale

```
$A = 1;
```

```
$B = "2";
```

```
$C = ($A + $B); // Somma di interi
```

```
$D = $A.$B; // Concatenazione di stringhe
```


```
echo $C; // Stampa 3
```

```
echo $D; // Stampa "12"
```

```
$4F = "ciao"; //Errore nel nome della var
```

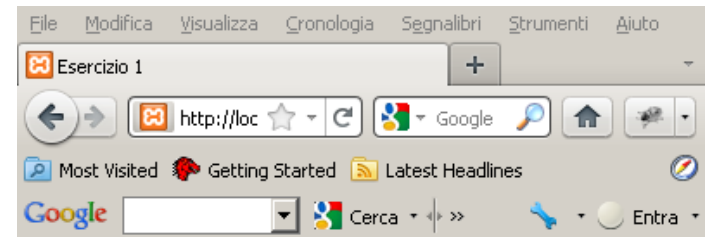
Primo programma

```
<html>
  <head>
    <title>Esercizio 1</title></head>
  <body>
    <h1>Esempio con PHP</h1>
    <p>
      <?php
        $a = "prova";
        $b = 18;
        $c = $b+77;
        echo "Questa &egrave; una $a
      </p>";
        echo "<p>Valore di c:$c </p>";
      ?>
    </body>
  </html>
```



Risultato

```
<html>
  <head>
    <title>Esercizio 1</title></head>
  <body>
    <h1>Esempio con PHP</h1>
    <p>
      <?php
        $a = "prova";
        $b = 18;
        $c = $b+77;
        echo "Questa &egrave; una $a </p>";
        echo "<p>Valore di c:$c </p>";
      ?>
    </p>
  </body>
</html>
```



Esempio con PHP

Questa è una prova

Valore di c:95

Ancora sulle variabili

- ❑ La function `isset()` controlla che la variabile argomento sia definita (**true**) o meno (**false**)
- ❑ Si può fare riferimento ad una variabile in modo indiretto

```
$prova='uno';
```

```
$$prova=3; //assegna 3 a $uno
```

```
$rosso=& $giallo /*entrambe le var. fanno  
rif. alla stessa area di memoria */
```

In realtà una variabile è semplicemente un riferimento tramite un nome ad una zona della memoria

Distruzione delle variabili

- ❑ Di solito non serve
- ❑ Si può usare `unset()`

```
$rosso=& $giallo;
```

```
unset($giallo);    // non distruggo $rosso
```

Altri operatori

❑ @

- ❖ Quando precede uno statement, gli eventuali errori generati vengono ignorati

❑ ` (backtick)

- ❖ La stringa all'interno viene passata allo shell ed il risultato è quello dell'esecuzione del comando

```
$output=`dir`;
```

```
echo $output;
```

Cast esplicito

❑ PHP ha un meccanismo di casting implicito ma supporta anche quello esplicito (sintassi tipo C)

❑ Esempio:

```
<?php
```

```
$a = 56;
```

```
$b = 12;
```

```
$c = $a/$b; // $c è 4.66666 (float)
```

```
$d = (int) ($a/$b); // $d è 4 (int)
```

```
?>
```

Statement if

- ❑ `if (condizione)`
 `istruzione;`
- ❑ `if (condizione)`
 `istruzione1;`
 `else`
 `istruzione2;`
- ❑ `if (condizione1)`
 `istruzione1;`
 `elseif (condizione2)`
 `istruzione2;`
 `...`
 `else`
 `istruzione3;`

Cicli

- `while (condizione)`
 `istruzione;`

- `do`
 `istruzione`
 `while (condizione) ;`

- `for (inizial. ; condizione ; incremento)`
 `istruzione;`

Switch

```
❑ switch ($x)
{
    case value1: istruzione1;
                break;
    case value2: istruzione2;
                break;
    ...
    default: istruzione3;
}
```

I valori possono essere numeri o stringhe, il confronto è fatto con == (loosely typed)

Sintassi alternativa

□ La { di un blocco è sostituita da :

□ La } di un blocco è sostituita da:

`endif; endwhile; endfor; endforeach;
endswitch;`

□ Esempio:

```
<?php
if ($a == 5) :
    echo "a equals 5";
    echo "...";
else:
    echo «a is not 5";
endif;
?>
```


Mix di controllo e HTML

□ Il codice PHP (anche singoli statements) possono essere spalmati su più processing instructions con codice HTML nel mezzo.

□ Esempio:

```
<?php if ($expression == true): ?>
```

This will show if the expression is true.

```
<?php else: ?>
```

Othersize this will show.

```
<?php endif; ?>
```

Mix di controllo e HTML

□ Il codice PHP (anche singoli statements) possono essere spalmati su più processing instructions con codice HTML nel mezzo.

□ Esempio:

```
<?php if ($expression == true): ?>
```

This will show if the expression is true.

```
<?php else: ?>
```

Othersize this will show.

```
<?php endif; ?>
```

Funzioni definite dall'utente

❑ Formato tipo sintassi C

```
function nome( lista param. )  
{ istruzioni }
```

❑ La lista parametri è opzionale ed usa come separatore la virgola

❑ Il nome della funzione è case **insensitive**

❑ La lista dei parametri sono variabili (che iniziano per \$)

❑ Esempio:

```
function sumsquares($a,$b)  
{ return ($a*$a+$b*$b) ; }
```

Chiamate di funzione

- ❑ Formato tipo sintassi C

`nome(lista param.)`

- ❑ Esempio:

`echo sumsquares(2,3); // prints 13`

- ❑ Quando chiamata, la funzione deve essere già stata processata dall'interprete PHP
- ❑ Si possono chiamare molte funzioni predefinite
- ❑ Si possono fare chiamate con numero variabile di parametri

Numero variabile di argomenti

- ❑ Per accedere ai parametro di una funzione (oltre ai parametri formali) vi sono le funzioni:
 - ❖ `$a=func_get_args()` //array di arg.
 - ❖ `$b=func_get_num()` //num. di arg.
 - ❖ `$val=func_get_arg[$i]` //valore arg. `$i`
- ❑ Il risultato di queste funzioni non può essere usato direttamente in espressioni, deve essere assegnato a delle variabili

```
function foo()  
{ $numargs = func_num_args();  
echo "Number of arguments: $numargs\n"; }
```

Regole di scoping

- ❑ Variabili definite dentro la funzione
 - ❖ Local scope : dentro la funzione nella quale è definita (gli argomenti della funzione rientrano in tale categoria)
- ❑ Variabili definite al di fuori delle funzioni
 - ❖ Scope: tutto il codice al di fuori delle funzioni (non visibili dentro le funzioni)
- ❑ Variabili globali dentro le funzioni
 - ❖ Una variabile dichiarata dentro una funzione con il qualificatore `global` ha scope globale
- ❑ Variabili statiche
 - ❖ Definite dentro una funzione tramite `static` sono come variabili locali ma mantengono il valore quando la funzione termina (come in C)

Esempio

```
$acc = 0;  
function sum($x) {  
    global $acc;  
    $acc += $x;  
}  
sum(10); sum(10);  
echo $acc;    // prints 20
```

Variabili «superglobal»

❑ Variabili globali predefinite

- ❖ Sono visibili e accessibili dovunque
- ❖ Hanno la forma di array associativi
- ❖ Tipicamente usate per informazioni di ambiente

❑ Esempi:

- ❖ `$GLOBALS` tutte le variabili globali correntemente definite nello script
- ❖ `$_GET` tutte le variabili passate allo script via HTTP GET
- ❖ `$_POST` tutte le variabili passate allo script via HTTP POST

- ❖ `$name = $_GET['name'] ;`

Passaggio parametri

- ❑ I parametri possono essere passati per valore o indirizzo
- ❑ Il default è il passaggio per valore
- ❑ I parametri passati per indirizzo sono preceduti da **&**
- ❑ Esempio:

```
Function add_some_extra(&$string) {  
    $string .= 'and something extra.'  
}  
$str = 'A string, '  
add_some_extra($str);  
echo $str; // prints 'A string, and something extra.'
```

Risultato «by reference»

- ❑ Il risultato (return) può essere passato per reference o per valore (default = valore)
 - ❖ Se il nome della funzione è preceduto da &, la funzione ritorna un reference

❑ Esempio:

```
function &get_x_ref() {  
    static $x = 30;  
    return $x;  
}  
  
$y = &get_x_ref(); // now $y is an alias for $x  
echo $y; // prints 30
```

Argomenti con valore di default

- ❑ Un valore di default può essere specificato per ogni argomento (come un assegnamento)
- ❑ Esempio:

```
function conc($a, $b, $sep=', '){  
    return ($a.$sep.$b);  
}  
  
echo conc("First", "Second", ', ');  
echo conc("First", "Second");  
  
// entrambi stampano la stessa cosa
```

Mettere funzioni in una variabile

- ❑ Una variabile che contiene il nome di una funzione può essere usata per invocare la funzione stessa

- ❑ Esempio:

```
function average($a,$b,$c) {  
    return ($a+$b+$c)/3;  
}  
$av="average";  
echo $av(1,2,3); // prints 2
```

Funzioni anonime

- Una funzione anonima può essere creata chiamando la funzione `create_function`
- Esempio:

```
$lambda=create_function('$a,$b,$c',  
    'return ($a+$b+$c)/3');  
echo $lambda(1,2,3); // prints 2  
  
// parameters  
// body
```

Terminazione dello script

- ❑ Le funzioni `exit()` , `die()` terminano l'esecuzione dello script
 - ❖ Possono prendere una stringa o un intero come parametri
 - ❖ La stringa è stampata prima della terminazione
 - ❖ L'intero è ritornato
- ❑ Esempio:

```
exit("connection failed");
```

Eccezioni

- ❑ Modello simile a quello di altri linguaggi (es. Java)
- ❑ Statement **throw** per segnalare una eccezione
- ❑ Statement **try . . . catch** per individuare e trattare le eccezioni

Esempio

```
<?php
function nfatt($x) {
    if (!is_int($x)) {
        throw new Exception('Operando non intero.\n');}
    elseif ($x<0){
        throw new Exception('Operando negativo.\n')}
    elseif ($x>13){
        throw new Exception('Operando troppo grande' . '\n')}
    else {
        for ($s=1;$x>0;$x--)
            $s*=$x;
        return $s;}
} //fine function nfatt

.....
try {
    echo nfatt(5) . "\n";
    echo nfatt(3.5) . "\n";
} catch (Exception $e) {
    echo 'Eccezione: ', $e->getMessage();
}
```

classe standard



Gestione implicita delle eccezioni

- ❑ Si può definire una funzione di trattamento errori non trattati da statement **catch**
- ❑ Si usa la seguente function
`set_exception_handler("nome funct.")`
- ❑ Se viene eseguito un throw senza catch, viene chiamata la funzione indicata nel param.
- ❑ Alla fine dell'esecuzione dell'handler, l'esecuzione **termina**

Esempio

```
<?php
function exception_handler($exception) {
    echo "Eccezione non catturata: ",
        $exception->getMessage(), "\n";
} // fine handler
.....
set_exception_handler('exception_handler');
.....
throw new Exception('Eccezione non prevista');

echo "Parte non eseguita\n";
?>
```

Arrays

- ❑ I vettori in PHP possono contenere dati di tipo **eterogeneo**
 - ❖ I vettori sono indirizzabili sia con un indice numerico che con un'etichetta di testo (array associativo)
- ❑ Creazione:
 - ❖ esplicita: `vettore=array(7,1967,"carciofi",12)`
 - ❖ implicita: `$vettore[0]=7; $vettore[1]=1967; ...`
 - ❖ associativa: `$t['nome']="Barney"; $t['cogn']="Gamble";
$t=array('nome'=>"Barney", 'cogn'=>'Gamble');`
- ❑ Accesso ai componenti:
 - ❖ `$vettore[$i]=... ; $tabella['nome']='Homer'; ...`

Esempio 2

```
<?php
$passwords=array(
    'bart' => 'calzino',
    'homer' => 'birra',
    'lisa' => 'nobel',
    'marge' => 'caspiterina',
    'maggie' => '' );
$numprimi[0] = 1;$numprimi[1] = 2;$numprimi[2] = 3;
?>
<html>
    <head><title>Esempio 2</title></head>
    <body>
        <h1>Array</h1>
        <?php
            $indice=2; $nome="homer";
            echo "<p>Password di $nome:
                    $passwords[$nome]</p>";
            echo "<p>Il primo in posizione $index:
                    $numprimi[$indice]</p>";
        ?>
    </body>
</html>
```

example2.php

Esempio

example3.php

```
<?php
$ingredienti[] = "uova";
$ingredienti[] = "sale";
$ingredienti[2] = "farina";
$ingredienti[31] = "miele";
echo "<P> Il vettore &grave; lungo ",
    count($ingredienti), "</P>";
echo "<UL> \n " ;
$c=count($ingredienti);
for($i=0; $i< $c; $i++) {
echo "<LI> ", $i+1;
echo " ",ucwords($ingredienti[$i]), " <br>"; }
// ucwords = uppercase , $ingredienti[3] ERRORE

echo "</UL> <BR> Trentaduesimo elemento:
    ",$ingredienti[31], "<br>";
echo "Centunesimo ingrediente: ", $ingredienti[100];
?>
```

Mozilla Firefox

File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto

Crui: Conferenza dei Rettori delle Univer... x http://localhost/prova/prova.php x +

http://localhost/prova/prova.php ☆ ↻ Google

Most Visited Getting Started Latest Headlines

Google Cerca Condividi Segnalibri Entra

Il vettore è lungo 4

- 1 Uova
- 2 Sale
- 3 Farina
- 4

(!) Notice: Undefined offset: 3 in C:\xampp\htdocs\prova\prova.php on line 14

Call Stack

#	Time	Memory	Function	Location
1	0.0005	337136	{main}()	..\prova.php:0

Trentaduesimo elemento: miele

Centunesimo ingrediente:

(!) Notice: Undefined offset: 100 in C:\xampp\htdocs\prova\prova.php on line 16

Call Stack

#	Time	Memory	Function	Location
1	0.0005	337136	{main}()	..\prova.php:0

Accesso alle chiavi

- ❑ Si può non conoscere i valori degli indici in un array associativo
- ❑ Si può usare la function `each ($nome_array)`
- ❑ Esempio

```
$a=array ( 'nome'=>"Mario" , 'cogn'=>'Rossi' ,  
    'nato'=>"13-4-1978") ;  
  
$c=count ($a) ;  
for ($i=0;$i<$c; $i++)  
{  
    $elem=each ($a) ;  
    echo $elem['key'] , " " , $elem["value"] , "  
    <br>\n" ;  
}
```

Funzione each ()

- ❑ In PHP, l'interprete ricorda quale è l'elemento corrente
- ❑ **each ()** produce un array associativo di 2 elementi con indici **value** e **key**
- ❑ Riempie l'array con il valore e chiave correnti
- ❑ Incrementa il puntatore corrente al successivo

Altre funzioni per iterazione

❑ **current()**

❖ Restituisce il valore dell'elemento corrente

❑ **key()**

❖ Restituisce il valore della chiave dell'elemento corrente

❑ **reset()** , **end()**

❖ Sposta il puntatore al primo/ultimo elemento

❑ **next()** , **prev()**

❖ Sposta il puntatore al prossimo/precedente elemento

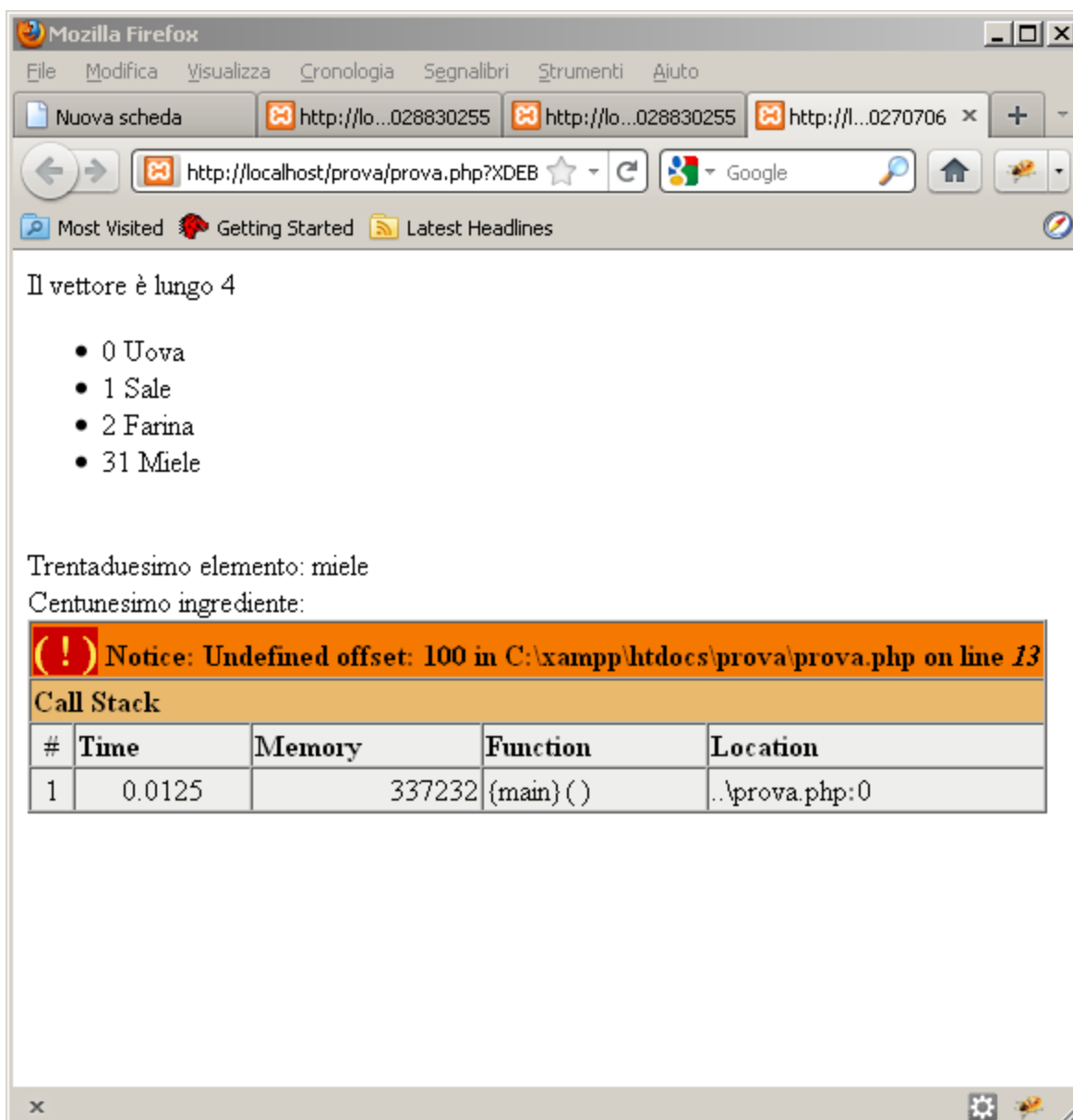
Cicli foreach

- ❑ Permettono di iterare sugli elementi di un array
- ❑ `foreach($nome_array as $contatore)`
 `istruzioni;`
- ❑ `$contatore` assumerà tutti i valori contenuti nell'array
- ❑ `foreach($vettore as $chiave=>$valore)`
 `istruzioni;`
- ❑ `$chiave $valore` esploreranno tutte le chiavi e valori nell'array

Stampa ingredienti senza (quasi) errori

```
<?php
$ingredienti[] = "uova";
$ingredienti[] = "sale";
$ingredienti[2] = "farina";
$ingredienti[31] = "miele";
echo "<P> Il vettore &grave; lungo ",
    count($ingredienti), "</P>";
echo "<UL> \n " ;
$c=count($ingredienti);
foreach ($ingredienti as $chiave=>$valore){
echo "<LI> ", $chiave;
echo " ",ucwords($valore), " <br>"; }
echo "</UL> <BR> Trentaduesimo elemento:
    ",$ingredienti[31], "<br>";
echo "Centunesimo ingrediente: ", $ingredienti[100];
?>
```

Risultato



The screenshot shows a Mozilla Firefox browser window with the address bar set to `http://localhost/prova/prova.php?XDEB`. The page content displays the following:

Il vettore è lungo 4

- 0 Uova
- 1 Sale
- 2 Farina
- 31 Miele

Trentaduesimo elemento: miele
Centunesimo ingrediente:

(!) Notice: Undefined offset: 100 in C:\xampp\htdocs\prova\prova.php on line 13

Call Stack

#	Time	Memory	Function	Location
1	0.0125	337232	{main} ()	..\prova.php:0

Principali funzioni per gli array

- ❑ **is_array()** ritorna true se il parametro è un array
- ❑ **count()** ritorna il numero di elementi nell'array
- ❑ **sort()** ordina l'array. Altri parametri opzionali specificano come ordinare.
- ❑ **explode()** , **compact()** creano un array da: stringhe (spezzettandole) o nomi di variabili (compact)
- ❑ **extract()** crea variabili da un array

Interazione con ambiente lato server

Interazione con le richieste

- ❑ Gli script PHP vengono attivati quando arriva una richiesta diretta alla risorsa corrispondente (file .php)
- ❑ Si deve poter interagire con
 - ❖ Ambiente (Environment)
 - ❖ GET
 - ❖ POST
 - ❖ Cookie
 - ❖ Server(cosiddetto EGPCS)

Variabili predefinite

□ Il server crea 6 array associativi per le informazioni EGPCS

- ❖ `$_ENV` valori delle variabili di ambiente
- ❖ `$_GET` valori passati da un modulo
- ❖ `$_POST` valori passati da un modulo
- ❖ `$_COOKIE` eventuali cookies nella richiesta
- ❖ `$_SERVER` informazioni sul server
- ❖ `$_FILE` informazioni su eventuali files trasmessi nella richiesta

□ Se si vuole sapere la struttura di queste variabili si può usare

```
var_dump(variable) o print_r(variable)
```


Stampa degli elementi

```
<h2 style="text-align:center; color:red;">
Stampa Variabili $_SERVER</h2>
<table border cellpadding=2>
<tr> <th> Variabile </th> <th> Valore </th>
  </tr>
<?php
  foreach($_SERVER as $indice =>$valore) {
    echo "<tr><td>$indice</td>
    <td>$valore</td>\n";
  }
?>
</table>
```

table.php

Esercizio 3

- ❑ Modificare l'istruzione dell'esempio 2 cosicché prenda indice e nome da un form HTML:

```
$indice=2; $nome="homer";
```

diventa:

```
$indice=$_GET['indice'];
```

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

- ❑ Introduciamo un file HTML con un form per trasmettere le informazioni

```
exercise2.php  
exercise2.html
```

```

<?php
$passwords=array(
    'bart' => 'calzino',
    'homer' => 'birra',
    'lisa' => 'nobel',
    'marge' => 'caspiterina',
    'maggie' => '' );
$numprimi[0] = 1;$numprimi[1] = 2;$numprimi[2] = 3;
?>
<html>
    <head><title>Esempio 2</title></head>
    <body>
        <h1>Array</h1>
        <?php
            $indice=$_GET['indice']; $nome=$_GET['nome'];
            echo "<p>Password di $nome:
            $passwords[$nome]</p>";
            echo "<p>Il $indice numero primo:
                $numprimi[$indice]</p>";
        ?>
    </body>
</html>

```

Risposta a GET e POST

- ❑ I dati di un form possono essere inviati sia tramite GET sia tramite POST
- ❑ Come conoscere quale metodo è stato usato?
 - ❖ Controllare entrambe le variabili
 - ❖ Usare la variabile per conoscere il metodo:
`$_SERVER['REQUEST_METHOD']`
 - ❖ Usare l'array `$_REQUEST` che include i valori di `$_GET`, `$_POST`, `$_COOKIE`

Revisione esercizio

```
if (isset($_GET['index']))  
    $index=$_GET['index'];  
else  
    $index=$_POST['index'];
```

```
switch($_SERVER['REQUEST_METHOD']) {  
    case 'GET': $index=$_GET['index'];  
                break;  
    case 'POST': $index=$_POST['index'];  
                break;  
}
```

```
$index = $_REQUEST['index'];
```

Funzione header

- ❑ Permette di scrivere una intestazione nella risposta
- ❑ **Attenzione!!** deve precedere ogni altro tipo di output

`<html>` ← **output precedenti**
`<body>` ← **output precedenti**
`<?php header (.....) ;` **errore!!**

❑ Sintassi

```
header( $string [, bool $replace = true  
        [,int $http_response_code ]] )
```


Accesso protetto

- ❑ Si possono utilizzare meccanismi standard dei server/browser (pop-up di una finestra sul browser)
 - ❖ Basic HTTP authentication
 - ❖ Digest HTTP authentication
- ❑ Per leggere i valori inseriti si usano le variabili
 - ❖ `$_SERVER['PHP_AUTH_USER']`
 - ❖ `$_SERVER['PHP_AUTH_PW']`
 - ❖ `$_SERVER['AUTH_TYPE']`
- ❑ Si può anche costruire un meccanismo proprio
- ❑ Per non autenticare ad ogni richiesta, conviene usare le **sessioni** (maggiori dettagli dopo)

Esempio

```
<?php
function verify($a,$b)
{return($a== "user" && $b==13);};
if(!isset($_SERVER['PHP_AUTH_USER']) ||
    verify($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])
    ==false){
    header('WWW-Authenticate: Basic realm="MyRealm"');
    header('HTTP/1.1 401 Unauthorized');
    echo 'Text to appear if user hits cancel';
    exit;
}
else {
    echo 'Correctly authenticated';
}
?>
```

HTTP header syntax



authentication.php

Gestione dei form: un esercizio

- un meccanismo spesso utilizzato per la gestione di form semplici prevede che ci sia una sola pagina per il form e per la sua gestione,
 - ❖ una condizione verifica se le variabili sono settate,
 - ❖ se sì, vengono elaborate e costruita la risposta;
 - ❖ se no, viene restituito il form

Esercizio

- sviluppiamo una calcolatrice, costituita da
 - ❖ un form con tre controlli (due operandi e un'operazione)
 - ❖ una pagina per la risposta

Possibile soluzione

□ Struttura:

```
<h1>Calculator:</h1>
```

```
<?php
```

```
if(isset($_GET['op1']) && isset($_GET['op2']) &&  
    $_GET['op1']!="" && $_GET['op2']!="") {
```

```
// variables are set, process them
```

```
...
```

```
} else {
```

```
// variables are not set, show form
```

```
?>
```

```
... // HTML form here
```

```
<?php
```

```
} // end of else branch
```

```
?>
```

calculator.php

Esercizio: note

- ❑ La variabile `$_SERVER['PHP_SELF']` contiene il nome dello script;
- ❑ La variabile `$_SERVER['REMOTE_ADDR']` contiene l'indirizzo IP del server

Trattare i file caricati

- ❑ **Attenzione:** permettere a chiunque di caricare files può produrre inconvenienti
- ❑ I files caricati da un form con POST sono descritti nell 'array associativo `$_FILES`
- ❑ Esempio

```
<form action="http://myserver/files.php"
      enctype="multipart/form-data"
      method="post">
file <input type=file name=immagine>
<input type=submit value="INVIA">
</form>
```

Accesso al file

- ❑ La variabile `$_FILE['immagine']` contiene le informazioni sul file caricato (se è stato caricato)
- ❑ `$_FILE['immagine']` è un array associativo

Elementi \$_FILE['immagine']

- ❑ **name** filename originale (completo)
- ❑ **tmp_name** nome del file temporaneo locale
- ❑ **type** tipo MIME del file
- ❑ **size** dimensione in bytes del file
- ❑ **error** codice di errore
- ❑ Prima di leggere un file, è buona pratica controllare che:
 - ❖ il file non sia troppo grande
 - ❖ il file sia del tipo atteso
 - ❖ non vi siano stati errori

Esempio

□ Come salvare il file

```
$dest="./img/";//dir. immagini
$dest=$dest.basename($_FILE['image']
                                ['name']);
if(move_uploaded_file($_FILE['image']
                    ['tmp_name'],$dest))
    { echo "file uploaded";}
else
    { echo "errore nell'\uploading del
file";}
}
```

estrae solo il nome

Altro esempio

❑ Come controllare il tipo del file

❖ si può usare la function

```
stripos (pagliaio, ago[, offset=0])
```

`pagliaio` stringa in cui cercare

`ago` stringa da cercare

`offset` punto di partenza della ricerca (default 0)

❖ restituisce la posizione del primo `ago` trovato nel `pagliaio`, oppure falso

❖ Esempio

```
if (stripos ($_FILE['immagine']['type'],  
            'image/')===false) //non va bene.....
```

Iniezione di codice

- ❑ Tecnica adottata da utenti maliziosi per modificare il comportamento normale di un sito
- ❑ Consiste nell'inserire del codice HTML, JavaScript o PHP nei campi di un form
- ❑ Esempio

```
<body>
```

```
<p> Ciao <?php $_POST['nome']; ?> ,
```

```
<br> tu hai <?php $_POST['eta'];?> anni. </p>
```

```
</body>
```

- ❑ Un utente può forzare il server ad eseguire del codice PHP arbitrario semplicemente mettendolo nei campi del form

Ancora su iniezione

- ❑ Le cose possono peggiorare se si manipolano files
 - ❖ si rischia di cambiare i file delle pagine web
 - ❖ si rischia di cambiare il contenuto di un DB

Come ci si può difendere

- ❑ Si possono analizzare le stringhe immesse

- ❑ Può essere utile usare

`strip_tags (stringa[, tag_amm])`

- ❑ Ritorna una stringa ripulita eliminando tutti i tag HTML e PHP, elimina inoltre i caratteri NULL

- ❑ `tag_amm` è una stringa con tutti i tag che si vuole escludere dalla ripulitura

Esempio

```
<?php
    $text = '<p>Test paragraph.</p><!--Comment-->
            <a href="#fragment">Other text</a>';
    echo strip_tags($text);
    echo "\n";
    // Allow <p> and <a>
    echo strip_tags($text, '<p><a>');
?>
```

Risultato

Test paragraph. Other text

<p>Test paragraph.</p>

Other text

Un'altra utile funzione

- ❑ **htmlspecialchars()** converte tutti i caratteri speciali in entità HTML, per esempio converte `<` in `<`
- ❑ La differenza rispetto a **strip_tags** è che in questo caso i tokens sono convertiti invece di essere cancellati
- ❑ Conclusioni:
 - ❖ **TUTTO** l'input proveniente dall'esterno (es. utente) va **validato lato server** (ovviamente non è sufficiente validazione lato client es. javascript)

Accesso a file

- ❑ PHP include molte funzioni per questo scopo
 - ❖ Funzioni C-like:
 - fopen, fclose, fread, fwrite, fseek, ftell, feof, rewind, fgets, fputs, flock
 - ❖ Funzioni tipo shell:
 - copy, rename, unlink, file_exists, file (legge tutto il file)
- ❑ Molte di queste funzionano anche con
 - ❖ File remoti (via URLs)
 - ❖ Sockets
- ❑ L'accesso a file remoti può essere disabilitato nel file di configurazione php.ini

Funzioni principali per lettura

- ❑ **fopen** (*nome*, *modo* [, *include* [, *context*]])
 - ❖ *nome* stringa con il nome del file
 - ❖ *modo* stringa con il modo (simile al C)
 - ❖ *include* da mettere a 1 o TRUE se si vuole cercare anche nel direttorio specificato da `include_path`
 - ❖ *context* gruppo di variabili che specificano parametri per gli stream
- ❑ **fgets** (*handle* [, *lunghezza*])
 - ❖ Ritorna una stringa (la linea letta) o FALSE
 - ❖ Se *lunghezza* è specificata, legge solo fino a *lunghezza*-1 caratteri

Esempi

❑ Leggere un file linea per linea

```
<?php
```

```
$handle = @fopen("/tmp/inputfile.txt", "r");  
if ($handle) {  
    while (($buffer = fgets($handle, 4096)) !== false) {  
        echo $buffer;  
    }  
    if (!feof($handle)) {  
        echo "Error: unexpected fgets() fail\n";  
    }  
    fclose($handle);  
}
```

```
?>
```

Lettura di un file intero

- ❑ Leggere un file intero con una sola istruzione

`file (nome [, flag=0 [, context]])`

- ❖ Il contenuto del file viene letto e ritornato come un array di stringhe
- ❖ Ogni stringa include il fine linea (EOL), a meno che il flag `FILE_IGNORE_NEWLINE` sia specificato

Esempi

- ❑ Leggere e stampare un file di testo

```
<?php
    $lines = file('http://www.example.com/');
    foreach ($lines as $line_num => $line)
        {echo "Line #<b>{$line_num}</b> : ".
            htmlspecialchars($line) . "<br>\n";
        } ?>
```

- ❑ Leggere il file eliminando fine riga e righe vuote

```
<?php
    $trimmed = file('somefile.txt',
FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
?>
```

Altre funzioni

- ❑ **`fwrite(handle, stringa[, lung])`**
 - ❖ Scrive la stringa nel file
 - ❖ Se *lung* è specificato, scrive al max *lung* caratteri
 - ❖ Ritorna il numero di caratteri scritti o FALSE se non è possibile scrivere

Sessioni

- ❑ HTTP è un protocollo **stateless**, ogni richiesta è indipendente dalle precedenti
- ❑ Il meccanismo delle sessioni è stato introdotto per consentire al server di riconoscere richieste collegate (es. dallo stesso browser)
- ❑ Utilità delle sessioni (esempi)
 - ❖ Autenticazione
 - ❖ Preferenze utente
 - ❖ Carrello della spesa
 - ❖ ...

Implementazione delle sessioni

- Una sessione è iniziata dal server
 - ❖ Il server crea un ID di sessione univoco e mantiene informazioni di sessione associate con questo ID
- L'ID è comunicato al client
 - ❖ Il client include l'ID in ogni richiesta successiva che deve essere riconosciuta come appartenente alla sessione
- La sessione è terminata dal server
 - ❖ Può accadere all'esecuzione di un comando di logout o quando scade un timeout (la sessione è rimasta inattiva per un certo tempo)

Implementazione delle sessioni

- Due meccanismi principali per l'implementazione delle sessioni
 - ❖ L'ID è inviato dal server al client come un «cookie», solamente con la prima risposta. Il cookie è immagazzinato sul client, che automaticamente lo invia al server con ogni richiesta
 - ❖ L'ID è codificato dal server negli URLs che sono scritti in ogni risposta, così che il client lo invia automaticamente quando fa accesso a quegli URLs

Sessioni in PHP

- ❑ Entrambi i meccanismi sono supportati. Il meccanismo dei cookies è quello di default
- ❑ Le informazioni di sessione possono includere un numero arbitrario di variabili
- ❑ L'ID di sessione può essere scelto
 - ❖ Automaticamente dal sistema
 - ❖ Dal programmatore (ma non deve essere facilmente «indovinabile»)
- ❑ L'accesso agli oggetti delle sessioni è **thread-safe** (possibile impatto sulle prestazioni)

Sessioni con ID di sistema

- ❑ Bisogna usare la function

`session_start()`

- ❖ IMPORTANTE: la chiamata deve precedere qualsiasi output dello script
- ❑ Se una sessione esiste ne crea una, oppure recupera la sessione già esistente
- ❑ L'ID proviene (di default) da un **cookie** nella richiesta
- ❑ Si possono definire le variabili da salvare semplicemente aggiungendo elementi all'array

`$_SESSION`

dopo aver chiamato `session_start()`

Esempio

```
<?php
```

```
    session_start(); //crea o recup. sess.
```

```
    $_SESSION['bgcolor']='yellow';
```

```
    $_SESSION['time']=time();
```

```
    $a=$_SESSION['date']; //se esiste!
```

```
.....
```

```
?>
```

NB: Attenzione ai nomi delle variabili usate: se il server è unico e il browser non viene chiuso la sessione rimane la stessa anche se l'applicazione è diversa (server in lab)

Esempio: contare i contatti

```
<?php
    session_start();
    if (isset($_SESSION['count'])) {
        $i=$_SESSION['count'];
    } else {
        $i=0;
    }
?>
<html>
<head><title>Esempio di concorrenza</title></head>
<body>
<h1> questo &grave; il collegamento numero:</h1>
<p>
<?php
    echo $i;
    $_SESSION['count']=$i+1;
?>
</body>
</html>
```

concurrency.php

Commenti sull'esempio

- ❑ Le sessioni valgono fino al momento della chiusura del browser
- ❑ La sessione identifica il browser, non la macchina, né l'utente
- ❑ Le richieste accedono ai dati della sessione in **mutua esclusione**
 - ❖ problema di prestazioni
 - ❖ (verificare nell'esempio precedente introducendo un ritardo tra la lettura del contatore e il suo incremento)

Variante con sessione unica

```
<?php
    session_id("sessione-unica");    // ID impostato dal programmatore
    session_start();
    if (isset($_SESSION['count']))
        { $i=$_SESSION['count'];
          }
        else { $i=0;} ?>

<html>
<head><title>Esempio di concorrenza</title></head>
<body>
<h1> questo è il collegamento numero:</h1>
<p>
<?php
echo $i;
$_SESSION['count']=$i+1;
session_write_close();
if (isset($_REQUEST['time'])) {sleep($_REQUEST['time']);}
    else {sleep(20);};    // Consente di verificare la mutua esclusione
echo "<p>".session_id();
?>
</body>
</html>
```

Gestione a tempo della sessione

- ❑ I dati delle sessioni sono in appositi files sul server
 - ❖ Se non sono distrutte automaticamente dopo un certo tempo di inattività possono creare leakages
- ❑ In PHP le sessioni troppo inattive sono eliminate da un **garbage collector**
- ❑ La vita garantita di una sessione può essere controllata con la function

```
ini_set('session.gc_maxlifetime',  
1800);
```

↑
secondi

↑
parametro nel file php.ini

Altri problemi

- ❑ Tutti le sessioni sono memorizzate insieme
- ❑ Il tempo di vita può essere differente per diversi siti co-localizzati
- ❑ Si possono memorizzare cartelle diverse le sessioni di siti differenti
- ❑ Esempio:

```
/* imposta cartella per le sess. di questo sito */  
session_save_path('/tmp/mydir');  
/* imposta periodo di gc a 7 giorni*/  
ini_set('session.gc_maxlifetime', 7*24*3600);  
/*imposta durata e path nei cookie */  
session_set_cookie_params(1800, '/');  
session_start();
```

Gestione esplicita di inattività

```
<?php
session_start();
$t = time();
$diff=0;
if (isset($_SESSION['time'])){
    $t0 = $_SESSION['time'];
    $diff = $t - $t0;}//periodo di inattività
else $nuovo=true;
if ($nuovo ||$diff > 1800 ) {/* nuovo o con periodo di
                                inattività troppo lungo */
    session_unset(); // svuota la sessione
    session_destroy(); // distrugge la sessione
header('HTTP/1.1 307 temporary redirect');
header('Location: index.php?msg=SessionTimeOut');
    /* ri-dirige il cliente alla pag. di login*/
exit;
}
else {
$_SESSION['time'] = time(); /* aggiorna tempo ultima attivazione
    */
}
?>
```

sessions.php

Altre funzioni legate alle sessioni

□ `session_get_cookie_params()`

❖ restituisce un array associativo con indici

- `lifetime`
- `path`
- `domain`
- `secure`
- `httponly`

□ `session_id ([sid])`

❖ restituisce il valore corrente dell'id

❖ setta un nuovo id al valore del parametro (se usato), da usare prima di `session_start()`

«Include» di codice PHP

- ❑ E' possibile includere e interpretare il contenuto di un altro file tramite **include**
 - ❖ Esempio: **include 'file.php' ;**
- ❑ Particolarmente utile per porzioni di codice PHP che devono essere replicate in più pagine
 - ❖ Funzioni di utilità
 - ❖ Controllo dei diritti di accesso (es. presenza di sessioni, timeout delle sessioni, ecc.)
 - ❖ Usare questo meccanismo e non il «copia e incolla» che facilita errori e rende difficile la manutenzione del codice

Funzioni per controllo cache

- ❑ I cookies sono in una cache nel **cliente**
- ❑ **`session_cache_expire`**(*durata*)
 - ❖ imposta la durata di tutti i cookie della sessione
 - ❖ durata in minuti
- ❑ **`session_cache_delimiter`**(*tipo*)
 - ❖ imposta parametri per il cache-control di pagine HTML
 - ❖ alcuni valori
 - *public*, normale (anche da parte di proxies)
 - *nocache*, non si può mettere la risposta in cache
 - *private*, in cache private
 - *private_no_expire*, senza scadenza, in cache private

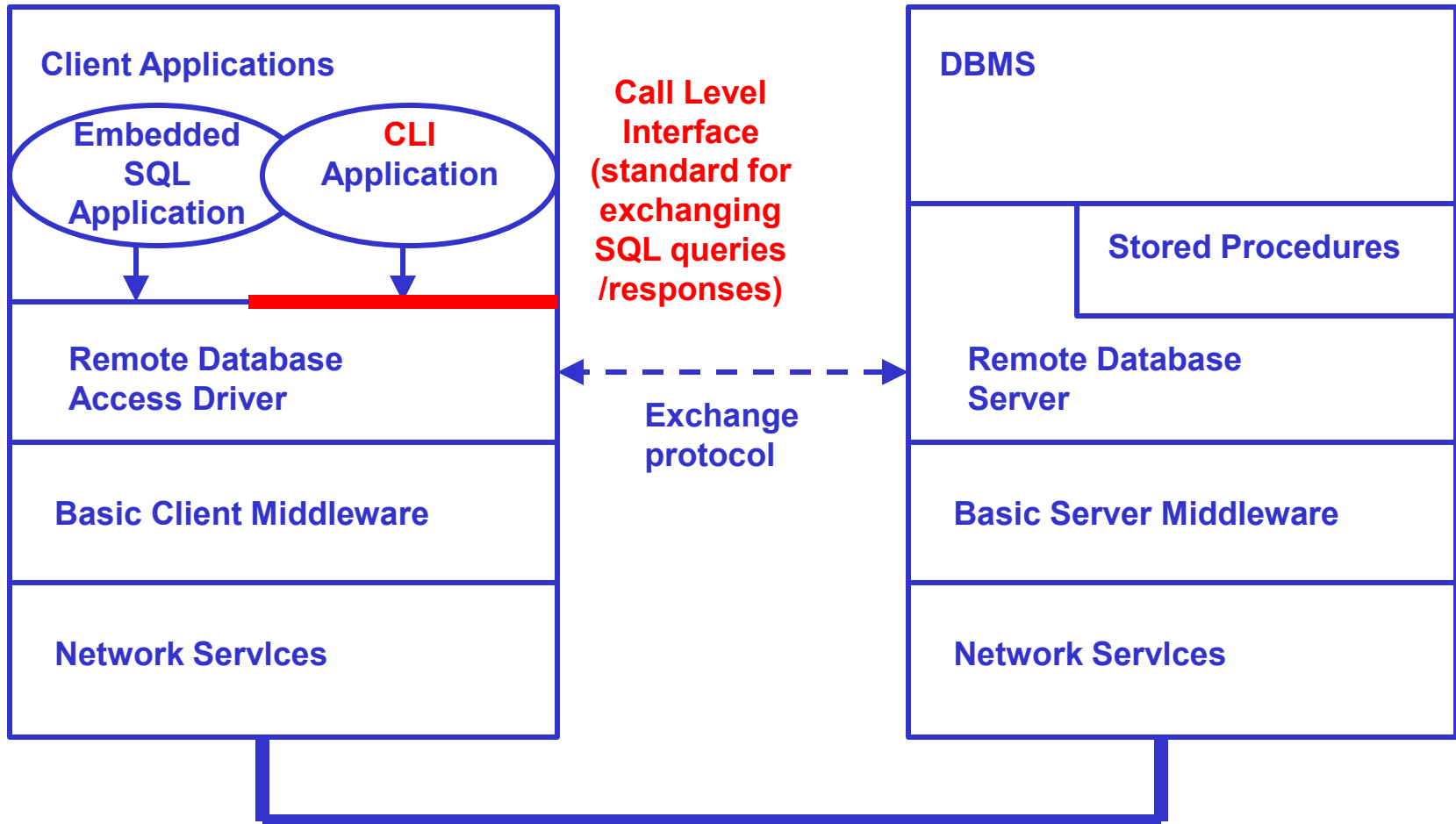
Accesso a basi di dati

- ❑ Tipicamente le soluzioni per database forniscono meccanismi di accesso remoto
- ❑ Obiettivo: dare accesso a dati distribuiti/replicati come se fossero una singola copia locale
- ❑ Problemi da risolvere:
 - ❖ Eterogeneità dei formati di dati e rappresentazioni
 - ❖ Accesso concorrente ai dati
 - ❖ Failures e recupero da crash
 - ❖ Sicurezza

Differenti tipi di servizi

- ❑ Single-site Remote Database Access (SRDA)
 - ❖ Accesso remoto a un singolo database (tipicamente fornito da un middleware proprietario specifico del singolo database)
- ❑ Distributed Query Processing (DQP)
 - ❖ Accesso remoto sola-lettura a database multipli (query su tabelle distribuite su più databases)
- ❑ Distributed Transaction Processing (DTP)
 - ❖ Transazioni distribuite su più database con possibilità di replicazione dei dati

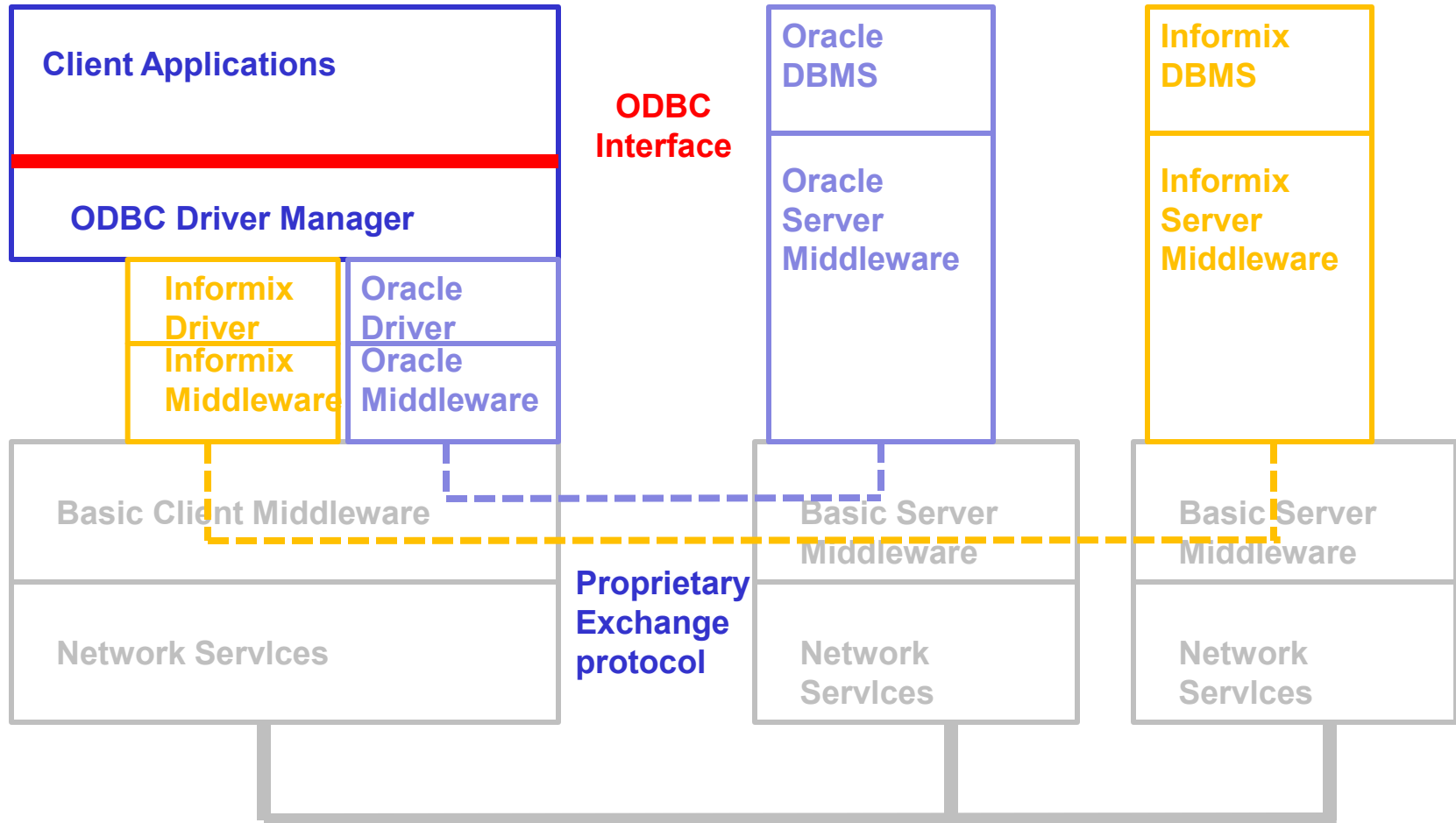
Tipica organizzazione client/server di SRDA SQL middleware



Standardizzazione delle API

- ❑ Gli sviluppatori di database forniscono i loro protocolli proprietari e le loro API
- ❑ La standardizzazione ha condotto a:
 - ❖ APIs aperte (es. ODBC e derivati)
 - ❖ Protocolli aperti (es. RDA)

Example: ODBC



Accesso a basi di dati

- ❑ PHP fornisce accesso sia tramite ODBC, sia tramite librerie specifiche per specifici software
- ❑ Nello specifico, vediamo MySQLI (I = improved)
 - ❖ prodotto molto diffuso, open source
 - ❖ basato sul linguaggio standard SQL
 - ❖ <http://www.mysql.com>
 - ❖ utilizza il protocollo binario per MySQL
 - ❖ Permette di avere transazioni e operazioni pre-compile

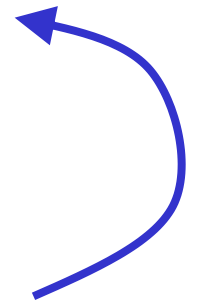
Libreria MySQLI

- ❑ Tecnicamente è un'estensione di PHP
- ❑ Molte funzioni (solo alcune sono presentate qui)
 - ❖ aprire e chiudere la connessione al DB server e selezionare il database;
 - ❖ inviare una istruzione SQL (che può essere una query, la creazione di un record, ecc.)
 - ❖ interpretare il risultato (può essere complesso, es. composto da decine di record con molti campi ciascuno):
 - ❖ operazioni predefinite («prepared statements») e transazioni
- ❑ Interfaccia procedurale o orientata ad oggetti
 - ❖ qui si presenta l'interfaccia procedurale

Uso della libreria mysqli

□ Procedura tipica

- ❖ Connessione al DB server
- ❖ Selezione del database da usare
- ❖ Preparazione della stringa SQL
- ❖ Invio della stringa SQL al DB server
- ❖ Recupero del risultato e
processamento/visualizzazione in pagina web
- ❖ Disconnessione dal DB server



Connessione

- ❑ Prima di eseguire qualsiasi operazione, bisogna stabilire una connessione
- ❑ **mysqli_connect(.....)** parametri (opzionali)
 - ❖ **host** ind. IP o nome macchina con DB, se manca è localhost, se preceduto da **p**: è connessione persistente
 - ❖ **username**
 - ❖ **passwd**
 - ❖ **dbname** nome DB default per le operazioni
 - ❖ **port** n. porta per il server MySQL
 - ❖ **socket** socket cliente da utilizzare

Esempio

```
<?php
    $conn = mysqli_connect('localhost',
                           'my_user', 'my_password', 'my_db');

    if (!$conn) {
        die('Errore nella connessione('
            . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
    }

    echo 'Successo'.mysqli_get_host_info($conn)."\n";

    ...

    mysqli_close($conn);
?>
```

Note

- ❑ Si possono aprire anche più connessioni contemporaneamente
- ❑ I risultati di `mysqli_connect_errno()` e `mysqli_connect_error()` si riferiscono all'ultima operazione tentata (interfaccia procedurale)
- ❑ Nell'interfaccia ad oggetti, ogni connessione ha i suoi attributi di errore

Connessioni persistenti e non

- ❑ L'operazione di connessione è alquanto lunga
- ❑ Connessioni persistenti consentono il re-uso (pooling)
- ❑ Prima di essere riusate è necessario:
 - ❖ rollback delle transazioni attive
 - ❖ tabelle temporanee chiuse e cancellate
 - ❖ unlock delle tabelle
 - ❖ reset delle variabili di sessione
 - ❖ statement preparati vengono chiusi
 - ❖ rilascio di lock effettuati con `get_lock`
 - ❖ chiusura handler

Selezionare un DB

- Da eseguirsi dopo la connessione

`bool mysqli_select_db(conn., nome_db)`

`conn.` id. di connessione precedentemente
ottenuto

`nome_db` stringa con il nome del DB

- Può servire per cambiare DB senza aprire una nuova connessione

Esempio

```
<?php
    $conn = @mysqli_connect("localhost",
        "my_user", "my_password", "test");
    if (mysqli_connect_errno()) {
        echo "Connessione fallita: ".
            mysqli_connect_error();
        exit();
    }
    ..... //operazioni su DB "test"
    mysqli_select_db($conn, "world");
    /*cambio DB su "world"*/
    ..... //operazioni su "world"
    mysqli_close($conn);
?>
```

Interrogazioni SQL (query)

`mixed mysqli_query (conn. , query[, modo])`

- ❖ *conn.* id. conness. precedentemente ottenuto
- ❖ *query* stringa con interrogazione SQL (select, show, describe, explain,.....)
- ❖ *modo*
 - `MYSQLI_USE_RESULT` risultato non in un buffer (grandi dimensioni)
 - `MYSQLI_STORE_RESULT` (default) risultato in un buffer
- ❖ Ritorna:
 - `FALSE` (error),
 - object se select, show, describe, explain
 - `TRUE` per altre queries

Esempio 1

```
<?php
$conn = @mysqli_connect("localhost",
                        "my_user", "my_password", "world");
if ($ris = @mysqli_query($conn,
                        "SELECT Name FROM progetti LIMIT 25"))
{
    echo "Select ha prod. un num. di righe
        pari a ". mysqli_num_rows($ris);
    ..... //altre operazioni su $ris permesse
    mysqli_free_result($ris); //libera buff.
}
else die ("Operazione non riuscita");
?>
```

Esempio 2

```
<?php
    $conn = @mysqli_connect("localhost",
                            "my_user", "my_password", "world");
    if ($ris = @mysqli_query($conn,
        "SELECT nome FROM progetti", MYSQLI_USE_RESULT))
    {
        ..... //altre operazioni su $ris
        ..... //non si possono fare altre op. sul DB

        mysqli_free_result($ris);
        //libera buffer ed accesso al DB
    }
?>
```

Estrarre dati di una query

- ❑ Mettere in un array la prossima riga di un risultato (NULL se siamo al fondo)

```
array mysqli_fetch_array(risultato,  
                        tipo)
```

risultato ottenuto prec. da una interrog.

tipo tipo di array

MYSQLI_ASSOC array associativo

MYSQLI_NUM array con indici numerici
 (nomi degli indici=nomi delle
 colonne della tabella)

MYSQLI BOTH entrambi (default)

Esempi

```
<?php
.....// operazioni di apertura e selezione DB
$query = "SELECT ID, Name FROM progetti WHERE
          citta='Roma' LIMIT 3";
$ris = @mysqli_query($conn,$query);
$riga = mysqli_fetch_array($ris,MYSQLI_NUM);
echo "<br>Primo progetto". $riga[0]." ".
                                     $riga[1];
$riga = mysqli_fetch_array($res,MYSQLI_ASSOC);
echo "<br>Ancora un progetto ".
      $riga["ID"].$riga["Name"]);
$riga = mysqli_fetch_array($res);
echo "<br>Terza volta ".$riga[0], $riga["Name"]);

mysqli_free_result($ris);
?>
```

Esempio: blog

- ❑ Si vuole costruire un sito web per un blog
- ❑ Il database utilizzato sarà costituito da due tabelle:
 - ❖ **post(id, data, titolo, testo)**, destinata a memorizzare i post pubblicati sul blog;
 - ❖ **utenti(utente, password)**, tabella di supporto, utilizzata per contenere i dati di autenticazione degli utenti

SQL per creare il DB

```
CREATE TABLE posts (  
    id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    data DATETIME NOT NULL,  
    titolo VARCHAR(255),  
    testo TEXT  
);
```

```
CREATE TABLE utenti (  
    utente VARCHAR(20),  
    password VARCHAR(32)  
);
```

```
INSERT INTO utenti VALUES ('CapitanBlogger',  
    md5('segreta'));
```



Immagazzina l'MD5 della password

Nota: memorizzare password

- ❑ L'ultima istruzione aggiunge un utente alla tabella degli utenti, indicando username e password
- ❑ La password però non viene memorizzata così come è, ma cifrata utilizzando l'algoritmo standard MD5
 - ❖ È una funzione in SQL ma esiste anche in PHP
- ❑ In questo modo nel DB non ci sono password in chiaro, ed aumenta la sicurezza perché anche chi ha accesso al DB non può conoscere le password

Implementiamo delle funzioni

□ Dobbiamo implementare:

- ❖ `registra($titolo, $testo)`
- ❖ `leggi($da, $quanti)`
- ❖ `numeroPost()`
- ❖ `utenteValido()`
- ❖ Più una funzione di supporto: `dbConnect()` ;

registra()

```
function registra($titolo, $testo)
{
    $conn = dbConnect();
    $titolo = nl2br(htmlentities($titolo));
    $testo = nl2br(htmlentities($testo));
    $data = date("Y-m-d H:i:s");

    $sql = "INSERT INTO posts(data, titolo, testo) VALUES('"
        . $data . "', '" . $titolo . "', '" . $testo . "')";
    if(!@mysqli_query($sql))
    {die("Errore nella query: ".$sql. "<br>". mysqli_error());}

    mysqli_close($conn);
}
```

convert `\n` in `
`

convert in characters **HTML** (es. `<` `>`)

leggi()

```
function leggi($da, $quanti = 10) {
    $conn = dbConnect();
    $risultato = array();
    $da=$da-1;
    $sql="SELECT * FROM posts ORDER BY data DESC LIMIT " .
        $da .", " . $quanti;
    if(! $risposta = @mysqli_query($sql))
        {die("Errore nella query: ".$sql. "<br>".
            mysqli_error()));
    while ($riga = mysqli_fetch_array($risposta)) {
        $risultato[] = $riga;
    }
    mysqli_close($conn);
    return $risultato;
}
```

numeroPost()

```
function numeroPost()  
{  
    $conn = dbConnect();  
    $sql = "SELECT MAX(id) as numero FROM  
posts";  
    if(!$risposta = @mysqli_query($conn,$sql))  
        {die("Errore nell'interrogazione: "  
            . $sql . "<br>" . mysql_error())};  
    $numero = mysqli_fetch_array($risposta);  
    mysqli_close($conn);  
    return $numero[0];  
}
```

utenteValido()

```
function utenteValido($utente, $password) {
    $conn = dbConnect();
    $sql = "SELECT password FROM utenti WHERE utente = '"
           . $utente . "'";
    if(! $risposta = @mysqli_query($conn,$sql))
        {die("Errore nella query: "
             . $sql . "<br>" . mysqli_error());}
    if (mysqli_num_rows($risposta) == 0)
        return FALSE;

    $riga = mysql_fetch_array($risposta);
    mysqli_close($conn);
    return (md5($password) == $riga[0]);
}
```

Alternativa

```
function utenteValido($utente, $password) {  
    $conn = dbConnect();  
    $sql = "SELECT * FROM utenti WHERE utente = '"  
        . $utente . "' AND password='" . md5($password) . "'";  
    if(! $risposta = @mysqli_query($conn,$sql))  
        {die("Errore nella query: "  
            . $sql . "<br>" . mysqli_error());}  
    if (mysqli_num_rows($risposta) == 0)  
        return FALSE;  
    else  
        return TRUE;  
}
```

cosa succede se la stringa \$utente non è quello che ci si aspetta?
(es. operatori logici ecc.)

Prevenire SQL Injection

```
❑ string mysqli_real_escape_string(
    connessione, stringa)
```

- ❖ **connessione** riferimento a connessione
- ❖ **stringa** stringa da trasformare
- ❖ trasforma in caratteri di escape i caratteri **NULL(0), LF, CR, \, ', " e ^z**

- ❑ usare gli statement pre-compilati, in cui i caratteri sono sempre interpretati per come sono scritti (non esistono delimitatori)

dbConnect()

```
function dbConnect() {  
    $conn = @mysqli_connect("localhost",  
                            "pwlsblog", "segreta"))  
    if(mysqli_connect_error())  
        {die("Errore nella connessione al db: "  
            .mysqli_connect_error())};  
    if(!@mysqli_select_db($conn,"pwlsblog"))  
        {die("Errore nella selezione del db: ".  
            mysqli_error())};  
    return $conn;  
}
```

Note

- ❑ Nella connessione possono accadere errori:
 - ❖ Un errore nella connessione al server (a cui passiamo uno username ed una password),
 - Es. server giù, utente non autorizzato o inesistente, utente non autorizzato da quella macchina
 - ❖ Un errore nella selezione del DB
 - DB non esistente, utente non autorizzato ad accedere a quel db

Transazioni

- ❑ Per default, la connessione è in modalità autocommit (commit automatico dopo ogni statement SQL). Per disabilitare il commit automatico si usa la function:

```
mysqli_autocommit($conn, false);
```

- ❑ Per eseguire manualmente il commit (o il rollback) si usano i comandi:

```
boolean mysqli_commit($conn)
```

```
boolean mysqli_rollback($conn)
```

Per poter essere eseguite devono essere supportate (altrimenti si genera un errore)

Tipica transazione

```
<?php
try {
    autocommit($conn, false);
    if(!@mysqli_query($conn, comando1))
        throw new Exception("comando1 fallito");
    if(!@mysqli_query($conn, comando2))
        throw new Exception("comando2 fallito");
    if(!@mysqli_query($conn, comando3))
        throw new Exception("comando3 fallito");

    .....
    mysqli_commit($conn);
} catch (Exception $e) {
    mysqli_rollback($conn);
    echo "Rollback ". $e->getMessage();
    .....
}
?>
```

Statement preparati

- ❑ Permettono di creare statement parametrizzati da eseguire con parametri attuali ogni volta diversi
- ❑ Bisogna:
 - ❖ inizializzare lo statement
 - ❖ associare una operazione SQL parametrizzata con lo statement
 - ❖ dare un valore al parametro
 - ❖ eseguire l'operazione
 - ❖ associare il risultato a delle variabili
 - ❖ caricare (fetch) del risultato
 - ❖ eventuale chiusura dello statement
- ❑ NB: Possono non essere disponibili in certe implementazioni (es. su server LABINF per esame)

Funzioni (1)

- ❑ `mysqli_stmt_init(connessione)`
 - ❖ restituisce un riferimento ad uno *statement*
- ❑ `mysqli_stmt_prepare(statement, SQL)`
 - ❖ *SQL* contiene una unica operazione SQL con parametri indicati da ?
 - ❖ restituisce un risultato booleano
- ❑ `mysqli_stmt_bind_param(statement, tipo, var1[, var2[, var3...]])`
 - ❖ associa le variabili date ai param. dello statement
 - ❖ *tipo* indica il tipo delle variabili "s" stringa, "i" intere, "b" blob e "d" double . Es: "sssi" = 3 stringhe e 1 intero
 - ❖ restituisce un booleano

Funzioni (2)

- ❑ `mysqli_stmt_execute(statement)`
 - ❖ esegue lo statement
- ❑ `mysqli_stmt_bind_result(var1[, var2[, var3...]])`
 - ❖ associa delle variabili alle colonne del risultato
 - ❖ restituisce un valore booleano
- ❑ `mysqli_stmt_fetch(statement)`
 - ❖ carica 1 nuova riga del risultato nelle var. scelte
 - ❖ restituisce NULL se non vi sono più righe
 - ❖ restituisce FALSE se vi è stato un errore

Funzioni (3)

- ❑ `mysqli_stmt_store_result(statement)`
 - ❖ memorizza in un buffer il risultato di uno stmt
- ❑ `mysqli_stmt_free_result(statement)`
 - ❖ libera il buffer con i risultati dello statement
- ❑ `mysqli_stmt_close(statement)`
 - ❖ chiude (distrugge) lo statement

Esempio

```
<?php
...// apertura connessione, ecc.
$stmt = mysqli_prepare($conn, "INSERT INTO Projects
                                VALUES (?, ?, ?, ?)");
mysqli_stmt_bind_param($stmt, 'sssd', $ID, $Name,
                                $City, $Value);

$ID = 'J33';
$Name = 'Aircraft';
$City = "Athens";
$Value = 11.2;
/* esegue il prepared statement */
mysqli_stmt_execute($stmt);
echo "Row inserted.\n";
    echo mysqli_stmt_affected_rows($stmt);

/* chiude lo statement */
mysqli_stmt_close($stmt);
?>
```

Esempio 2

```
<?php
...// apertura connessione
$query = "SELECT Name, City FROM Projects ORDER by
                                                ID DESC LIMIT 150,5";
if ($stmt = @mysqli_prepare($conn, $query)) {
    mysqli_stmt_execute($stmt); // esegue lo statement
    mysqli_stmt_bind_result($stmt, $name, $city);
    while (mysqli_stmt_fetch($stmt)) {
        echo $name." ".$city."<br>\n";
    }
    /* chiude lo statement */
    mysqli_stmt_close($stmt);
}
else {die('Errore nello statement preparato');}
/* chiude la connessione */
mysqli_close($conn);
?>
```

Esempio 3

```
<?php
...// apertura connessione
$query = "SELECT Name, City FROM Projects ORDER by
                                                ID DESC LIMIT 150,5";
if ($stmt =@mysqli_prepare($conn, $query)) {
    mysqli_stmt_execute($stmt); /*esegue lo
                                statement*/

    mysqli_stmt_store_result($stmt);
    mysqli_stmt_bind_result($stmt, $name, $city);
    while (mysqli_stmt_fetch($stmt)) {
        echo $name." ". $city."<br>\n";
    }
    mysqli_stmt_free_result($stmt);
    /* chiude lo statement */
    mysqli_stmt_close($stmt);
}
else {die("errore nello statement preparato");}
/* chiude la connessione */
mysqli_close($conn); ?>
```

Osservazioni

- ❑ I parametri non possono apparire ovunque nell'operazione SQL
 - ❖ non possono essere nomi di colonne
 - ❖ gli operandi di un confronto non possono essere entrambi parametri
 - ❖ ...
- ❑ I risultati memorizzati in un buffer non possono cambiare durante la lettura
- ❑ Si può liberare un buffer ed eseguire nuovamente lo statement

Oggetti in PHP

□ In PHP si possono definire delle classi

```
class contatto {  
    protected $nome, $tel, $note;  
    public function __construct($nome) metodo  
costruttore  
    { this->nome=$nome; }  
    public function set_tel($tel)  
    { this->tel=$tel; }  
    public function set_note($note)  
    { this->note=$note; }  
    public function stampa_contatto()  
    { $out="Nome: ".this->nome."<br>Tel.: ".  
        this->tel."<br>Note: ".this->note;  
        return $out; }  
}
```

Visibilità

- ❑ Ogni proprietà di una classe può essere preceduta da:
 - ❖ **public** indica che la proprietà è accessibile da tutti
 - ❖ **private** indica che la proprietà è accessibile solo dal codice della stessa classe in cui è definita
 - ❖ **protected** indica che la proprietà è accessibile solo dalla stessa classe in cui è definita, da quelle predecessori e da quelle derivate
- ❑ Se si usa **var** la variabile è pubblica

Creare oggetti

- ❑ Si possono creare istanze di una classe già definita
- ❑ Quando si crea l'istanza, viene invocato il metodo costruttore
`$contact=new contatto("Mario") ;`
- ❑ Il costruttore può anche avere lo stesso nome della classe (deprecato)
- ❑ Non è obbligatorio avere un costruttore
- ❑ Può esistere un metodo distruttore, chiamato `__destruct()`, invocato dal garbage collector

Ereditarietà

```
class contatto_lavoro extends contatto
{
    protected $azienda, $ruolo;
    public function __construct($a,$b)
    {
        contatto::__construct($a);
        this->ruolo=$b;
    }
    public function set_az($azienda)
    {
        this->azienda=$azienda;
    }
    public function stampa_contatto()
    {
        $out="Nome: ".this->nome."<br>Tel.: ".
            this->tel."<br>Note: ".this->note.
            "<br>Azienda: ".this->azienda.
            "<br>Ruolo: ".this->ruolo";
        return $out;
    }
}
```

nome classe

Esempio di utilizzo

```
$a=new contatto("Mario");  
$b=new contatto_lavoro("Giovanni",  
                        "Software Engineer");
```

```
$a->set_tel("+39011543789");
```

```
$b->set_az("ACME Gadgets");
```

```
echo $a->stampa_contatto();
```

```
echo $b->stampa_contatto();
```

Chiamate di
metodi

```
$b->fax="+390113874690";
```

proprietà definita
"al volo"

Altre caratteristiche

- ❑ La classe deve essere definita in un solo blocco `<?php.....?>`
- ❑ Si può ereditare da **una sola classe**
- ❑ Si può sfruttare nel costruttore la possibilità di chiamata con numero di parametri variabile
- ❑ Si possono definire **classi astratte** precedute da **abstract**
 - ❖ contengono metodi di cui si dà solo il prototipo
 - ❖ la definizione del metodo è lasciata alle classi derivate
 - ❖ metodi privati non possono essere astratti

Interfacce

□ E' una classe completamente astratta

❖ Keyword **interface**

```
interface Storable {function getContentsAsText();}
```

□ Le interfacce sono utili per separare completamente l'interfaccia dall'implementazione

Esempio

```
class Document implements Storable
{ public function getContentsAsText()
    { return "This is Text of the Document\n"; }
}
class Indexer {
public function readAndIndex(Storable $s) {
$textData = $s->getContentsAsText();
echo $textData; }
}
$p = new Document();
$i = new Indexer();
$i->readAndIndex($p);
```

Classi astratte vs interfacce

Classe astratta

- ❑ Può definire alcune funzioni e lasciarne altre alle classi derivate
- ❑ Le classi derivate possono o meno ridefinire i metodi
- ❑ Ci deve essere una relazione logica fra la classe base e quelle derivate

Interfaccia

- ❑ Non può contenere elementi procedurali
- ❑ La classe che implementa l'interfaccia **deve** definire i metodi relativi
- ❑ Le classi che implementano la stessa interfaccia possono essere scorrelate

Serializzazione

- Si può serializzare il contenuto di un oggetto (es. per salvarlo su un file)

```
$a=new contatto("Mario");  
$a->set_tel("+39011543789");  
$a->set_note("Da conservare");  
$str=serialize($a);  
/* $str viene scritto in un file */  
/* si rilegge dal file in $str */  
$b=unserialize($str); //ripristino
```

Magic functions

- ❑ Funzioni che quando ridefinite in una classe hanno un significato particolare. Il loro nome inizia per `__`
- ❑ Esempi:
 - ❖ `__clone()` definisce cosa fare quando si vuole duplicare un oggetto
 - ❖ `__get`, `__set` definiscono cosa fare (es. controlli) quando si dà un valore (anche "al volo") ad una variabile dell'oggetto o la si legge
 - ❖ `__call` definisce cosa fare quando si invoca un metodo definito "al volo"
 - ❖ `__sleep`, `__wakeup` definiscono cosa salvare di un oggetto che viene serializzato, e come ripristinarlo
 - ❖ `__isset`, `__unset` invocate quando si applica `isset` o `unset` ad una proprietà dell'oggetto
 - ❖

Esempio: accesso a MySQL DB con API ad oggetti

□ Classi principali:

- ❖ **mysqli** classe che rappresenta una connessione a MySQL, contiene tutti i metodi applicabili alle connessioni
- ❖ **mysqli_stmt** classe che definisce uno statement preparato
- ❖ **mysqli_result** classe che rappresenta un result set di una operazione su DB
- ❖

□ Per eseguire una operazione, si crea un oggetto oppure si invoca un metodo

Esempio

```
$conn=@new mysqli("localhost", $user, $passwd, $db) ;  
if (mysqli_connect_errno()) {  
die("Connect failed: <br>".  
                                mysqli_connect_error());  
} // apre e controlla connessione  
$stmt = $conn->prepare("INSERT INTO Projects  
                        VALUES (?, ?, ?, ?)");  
  
$ID = 'J33';  
$Name = 'Aircraft';  
$City = "Athens";  
$Value = 11.2;
```

Esempio (2)

```
$stmt->bind_param('sssd', $ID, $Name, $City, $Value) ;  
/* esegue il prepared statement */  
$stmt->execute() ;  
echo "Row inserted.<br>" ;  
echo $stmt->affected_rows() ;  
  
/* chiude lo statement */  
$stmt->close() ;  
$conn->close() ; //chiude la connessione
```

Altro esempio (1)

```
$conn = @new mysqli("localhost", $user, $passwd, "world");
if (mysqli_connect_errno()) {
    die("Connect failed: ".mysqli_connect_error());
}

$query = "SELECT Name, City FROM Projects
          ORDER by ID LIMIT 3";

if (!$result = @$conn->query($query))
    {die("Errore nell'interrogazione" . $query."<br>");}

/* numeric array */
if ($riga = @$result->fetch_array(MYSQLI_NUM))
    {echo $riga[0]." ". $riga[1]."<br>\n";}
```

Altro esempio (2)

```
/* array associativo */
if($riga = @$result->fetch_array(MYSQLI_ASSOC))
{echo $row["Name"]." ". $row["City"]."<br>\n";}

/* array associativo e numerico */
if($riga = @$result->fetch_array(MYSQLI_BOTH))
{ echo $riga[0]." ".$riga["City"]."<br>\n";}
/* libera il result set */
$result->close();

/* chiude la connessione */
$conn->close();
```