Introduzione a LAMP

Francesco P. Lovergine

Consiglio Nazionale delle Ricerche

Associazione LugBARI

<f.lovergine@ba.issia.cnr.it>

Sommario

Presentazione del linguaggio object-oriented PHP e della progettazione di applicazioni orientate al web con tale linguaggio e l'ausilio di un DBMS.

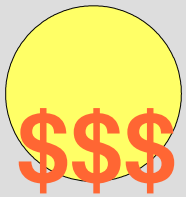
Note stilistiche sulla programmazione di applicazioni 3T di tipo WAMP o LAMP.

Altro...

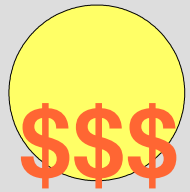
LAMP

- Una piattaforma di sviluppo specifica per la creazione di applicazioni orientate al web. Non proprietaria!

Linux + Apache + MySQL + PHP/Perl



IIS + ASP + MS SQL



PHP?

- Un quasi-acronimo per *Hypertextual Pre-Processor*.
- Attualmente alla versione **4.3.5** (versione 5 in sviluppo)
- Ben integrato con Apache, il webserver più diffuso in Internet (~ 65% dei server, fonte Netcraft)
- Linguaggio object-oriented e modulare, molto simile a C e quindi di facile apprendimento.

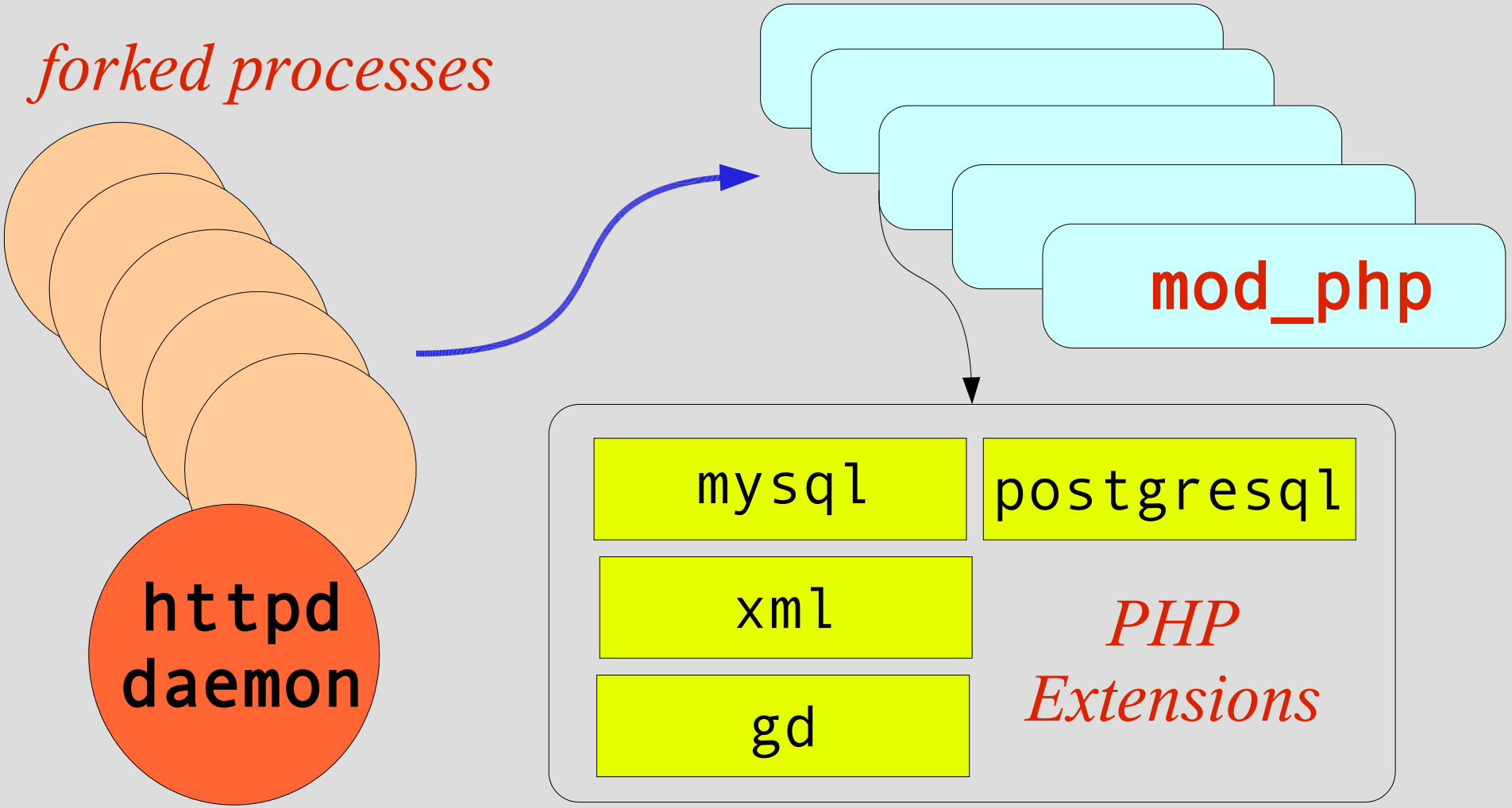
Vantaggi di PHP

- Semplice
- Efficiente (Zend)
- Object oriented, e quindi adatto alla realizzazione di interfacce ben strutturate
- Moltissimi moduli di interfaccia verso le più comuni librerie
- Molto ben documentato
- Aperto e a sviluppo distribuito (vendor independent)

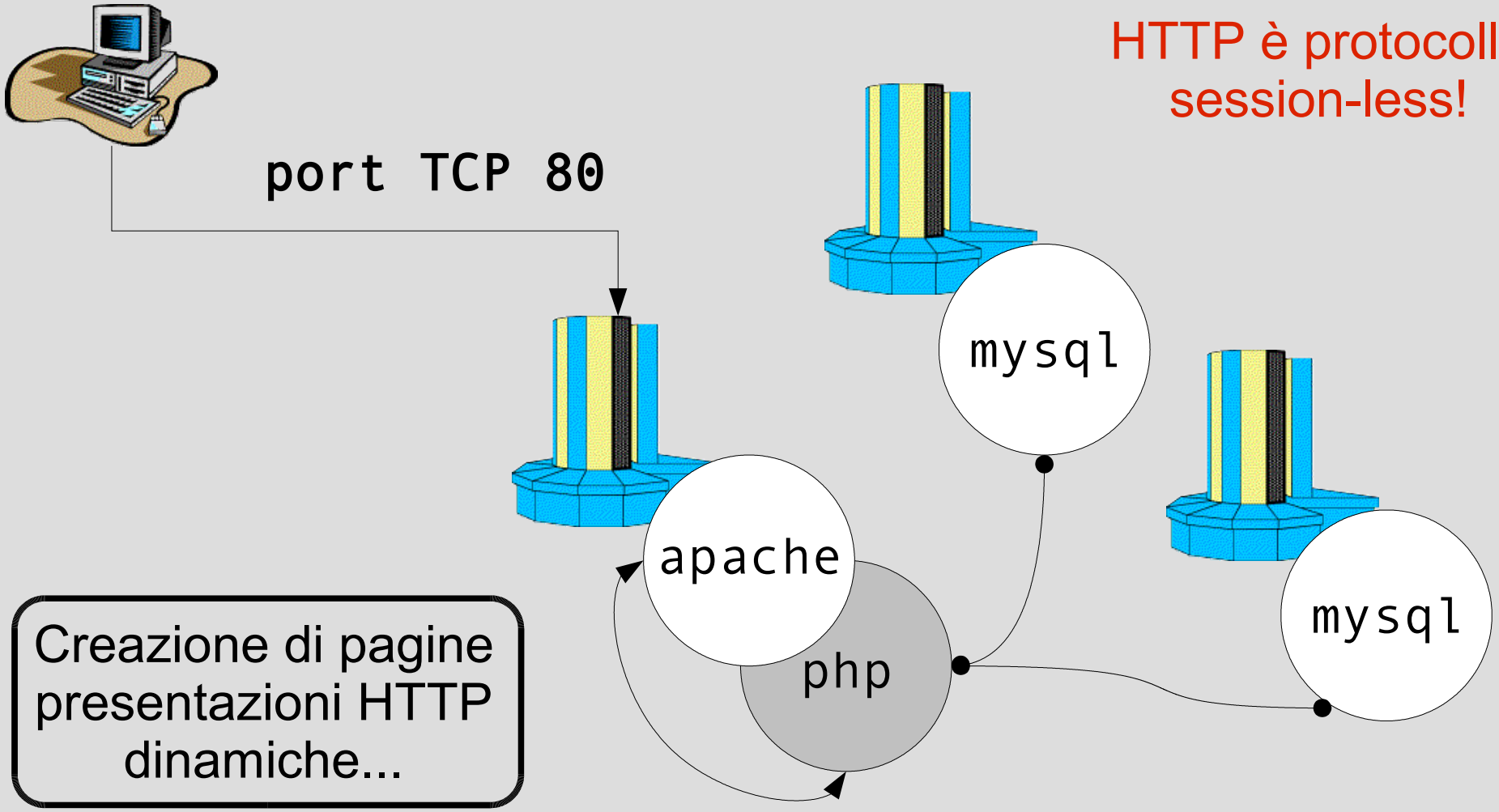
Svantaggi di PHP

- Si possono scrivere facilmente programmi difficili da manutenere per lo stesso autore
- Il mix di HTML e codice porta facilmente al caos se non si lavora in modo ordinato e stilisticamente corretto.
- Problemi di sicurezza subdoli – es. cross-site scripting
- Strutturazione dei dati non eccezionale

Architettura di Apache I



Architettura di Apache II



PHP

- E' un linguaggio interpretato che può essere immerso in HTML
- Generalmente viene eseguito come modulo di apache, ma è eseguibile anche come interprete *standalone*
- Nella prima modalità ha alcune limitazioni e diventa *specializzato*, ma è la configurazione usata più comunemente.
- Può essere eseguito come applicazione CGI (ugualmente poco usato)

MySQL

- E' un DBMS di tipo relazionale che implementa un dialetto di SQL99 come DML e DDL.
- Ha diverse limitazioni rispetto a più blasonati DBMS, ma è molto efficiente, soprattutto in lettura.
- E' il sistema di elezione per molte applicazioni orientate al web, che in genere non richiedono funzionalità avanzate...

Free software

- Una soluzione LAMP è tipicamente free software/open source.
- C'è la possibilità di adottare una licenza di tipo commerciale per entrambi i prodotti e rendere il codice opaco (vedi **ZEND**).
- Linux/Unix è la piattaforma che meglio si adatta a tali software, ma può essere utilizzato anche un sistema M$...

Le licenze free ≠ gratis

- Libertà di eseguire il programma, per qualsiasi scopo.
- Libertà di studiare come funziona il programma e adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito
- Libertà di ridistribuire copie in modo da aiutare il prossimo.
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

Un esempio...

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <?php
        echo "Hi, I'm a PHP script!";
        ?>
    </body>
</html>
```

E il risultato...

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        Hi, I'm a PHP script!
    </body>
</html>
```

Diversi modi di immergere codice in HTML

```
1. <?php
   echo("per scrivere documenti XHTML o XML fai così\n");
   ?>

2. <?
   echo ("Istruzione di processamento SGML\n");
   ?>
   <?= expression ?> sta per <? echo expression ?>"

3. <script language="php">
        echo ("some editors (like FrontPage) don't
              like processing instructions");
   </script>

4. <% echo ("You may optionally use ASP-style tags"); %>
   <%= $variable; # sta per "<% echo . . ." %>
```

Escaping più complesso

```
<?php
if ($expression) {
    ?>
    <strong>Questo è vero</strong>
    <?php
} else {
    ?>
    <strong>Questo è falso</strong>
    <?php
}
?>
```

Commenti

```
<?php
    echo "This is a test"; // This is a one-line c++
style comment
    /* This is a multi line comment
       yet another line of comment */
    echo "This is yet another test";
    echo "One Final Test"; # This is shell-style style
comment
?>
```

Accetta commenti in stile C, C++ e
Unix shell e il linguaggio è case-sensitive

Sommario PHP

- Variabili e tipi
- Costanti
- Espressioni e operatori
- Strutture di controllo
- Funzioni
- Classi e operatori sulle classi
- Feature del linguaggio e sicurezza
- Alcuni moduli utili ...

Variabili e tipi

- 4 tipi scalari: boolean, integer, float, string
- 2 tipi composti: array, object
- 2 tipi speciali: reference, NULL

Il linguaggio non è strettamente tipizzato.
Il tipo è deciso a run-time
context-based

float == double
settype() e cast permettono di forzare il tipo
Le variabili iniziano con $

Boolean

- TRUE o FALSE (non case sensitive)
- Cast: (bool) | (boolean)
- Si considerano valori FALSE:
  - integer 0
  - float 0.0
  - string “” oppure “0”
  - array con zero elementi
  - object con zero variabili
  - NULL

Integer

<?php
decimal : [1-9][0-9]* | 0

hexadecimal : 0[xX][0-9a-fA-F]+

octal : 0[0-7]+

integer : [+-]?*decimal* | [+-]?*hexadecimal* | [+-]?*octal*
?>

In caso di overflow la variabile viene promossa a float.
I bool corrispondono a 0 o 1 nelle espressioni.

Cast: `(int)` o `(integer)`

Float

```
LNUM            [0-9]+
DNUM            ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

La precisione dipende dalla piattaforma, ma per 64 bit
~1.8e308

Strings

```
<?php
echo 'this is a simple string';

echo 'You can also have embedded newlines in
strings this way as it is
okay to do';

// Outputs: Arnold once said: "I'll be back"
echo 'Arnold once said: "I\'ll be back"';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\\*.*?';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\*.*?';

// Outputs: This will not expand: \n a newline
echo 'This will not expand: \n a newline';

// Outputs: Variables do not $expand $either
echo 'Variables do not $expand $either';
?>
```

Single quote

Double quote – Interpolazione

Heredoc

```
echo <<<EOT
My name is "$name". I am printing some
$foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should print a capital 'A': \x41
EOT;
```

escaping

```
\\
\$
\r
\n
\t
\”
\[0-7]{1,3}
\x[0-9A-Fa-f]{1,2}
```

Ancora stringhe

```
// Get the first character of a string
$str = 'This is a test.';
$first = $str{0};

// Get the third character of a string
$third = $str{2};

// Get the last character of a string.
$str = 'This is still a test.';
$last = $str{strlen($str)-1};
```

Array

Sono equivalenti agli 'hash' o dizionari di altri linguaggi.

L'indice può quindi essere un integer o una stringa chiave.

Possono essere innestati e a multindice.

La dimensione non è predefinita e può essere estesa o ridotta a piacimento.

L'inizializzazione viene fatta con il token `array()` oppure assegnando semplicemente gli elementi.

Vediamo qualche esempio ...

Esempi di array

```
// this
$a = array( 'color' => 'red',
            'taste' => 'sweet',
            'shape' => 'round',
            'name'  => 'apple',
                       4        // key will be 0
          );

// is completely equivalent with
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // key will be 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// will result in the array array(0 => 'a' , 1 => 'b' , 2 => 'c'),
// or simply array('a', 'b', 'c')
```

Esempi di array

```
// Array as (property-)map
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// strictly numerical keys
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// this is the same as array(0 => 7, 1 => 8, ...)

$switching = array(         10, // key = 0
                     5    =>  6,
                     3    =>  7,
                     'a'  =>  4,
                              11, // key = 6 (maximum of integer-indices was 5)
                     '8'  =>  2, // key = 8 (integer!)
                     '02' => 77, // key = '02'
                     0    => 12  // the value 10 will be overwritten by 12
                   );

// empty array
$empty = array();
```

Esempi di array

```
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo "Do you like $color?\n";
}

/* output:
Do you like red?
Do you like blue?
Do you like green?
Do you like yellow?
*/
```

```
foreach ($colors as $key => $color) {
    // won't work:
    //$color = strtoupper($color);

    // works:
    $colors[$key] = strtoupper($color);
}
print_r($colors);

/* output:
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
    [3] => YELLOW
)
```

Scansione di un array con foreach() ...

Esempi di array

```
$fruits = array ( "fruits"  => array ( "a" => "orange",
                                       "b" => "banana",
                                       "c" => "apple"
                                     ),
                  "numbers" => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6
                                     ),
                  "holes"   => array (      "first",
                                       5 => "second",
                                            "third"
                                     )
                );

// Some examples to address values in the array above
echo $fruits["holes"][5];    // prints "second"
echo $fruits["fruits"]["a"]; // prints "orange"
unset($fruits["holes"][0]);  // remove "first"

// Create a new multi-dimensional array
$juices["apple"]["green"] = "good";
```

Multidimensionali

Objects

```
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();
```

Metodi + variabili

Importanza del contesto

```
$foo = "0";  // $foo is string (ASCII 48)
$foo += 2;   // $foo is now an integer (2)
$foo = $foo + 1.3;  // $foo is now a float (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";     // $foo is integer (15)

$obj = (object) 'ciao';
echo $obj->scalar;  // outputs 'ciao'
```

* (int), (integer) - cast to integer
* (bool), (boolean) - cast to boolean
* (float), (double), (real) - cast to float
* (string) - cast to string
* (array) - cast to array
* (object) - cast to object

Variabili

Le variabili hanno *scope* locale e vengono passate di default per valore a meno che non si usino le *reference* mediante operatore & Sono inoltre con allocazione *auto,* quindi non persistenti.

```
$foo = 'Bob';
$bar = &$foo;
$bar = "My name is $bar";
echo $bar;
echo $foo;
```

Al contrario del C, anche se definite al di fuori di funzioni e classi le variabili conservano la località...

Un esempio...

```
$a = 1; /* global scope */

function Test()
{
    echo $a; /* reference to local scope variable */
}

Test();
```

Qual è l'output del programma?

La versione corretta...

```
$a = 1;
$b = 2;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
}

Sum();
echo $b;
```

... oppure :-)

```
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo $b;
```

SUPERGLOBALS

Altri superglobals

$_SERVER

$_GET
$_POST
$_ENV
$_COOKIE
$_SESSION

$_REQUEST (== $_GET + $_POST + $_COOKIE)

Dalla versione 4.2 a seguire si devono sempre considerare register_globals in OFF.

ED E' COSA BUONA E GIUSTA!!!

Variabili static

```
function Test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
```

Sono ammesse le funzioni ricorsive

Variabili di variabili

```
$a  = "hello";
$$a = "world";

echo "$a ${$a}";
echo "$a $hello”;
```

In cosa sono diversi gli output?

Importare variabili HTTP

```
<form action="foo.php" method="POST">
    Name:  <input type="text" name="username"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" name="submit" value="Submit me!">
</form>
```

```
   print $_POST['username'];
   print $_REQUEST['username'];

   import_request_variables('p', 'p_');
   print $p_username;

//
// Disponibile in PHP3+ Potra' essere disabilitato in
// PHP5 con la direttiva register_long_arrays.
//
   print $HTTP_POST_VARS['username'];

   print $username;
```

Costanti

```
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
```

Le costanti sono globali, a sola lettura e distinguibili dalle variabili per il fatto che non iniziano con $

Non usate mai una variabile se è più appropriato usare una costante !

Costanti pre-definite classiche: __LINE__ e __FILE__

Operatori

```
Aritmetici: + - * / %
Assegnazioni: = += -= /= *= %= .= <<= >>=
Concatenazione: .
Bit: & | ~ ^ >> <<
Confronto: == === <= >= < > != <> !===
Esecuzione: `
Incremento/decremento: ++ --
Logici: and or xor ! && ||
Array: +
```

Controlli

```
if ( expr ){} else {}
if ( expr ) {} elsif (expr) {} else {}
while ( expr ) {}
do {} while ( expr );
for ( expr1; expr2; expr3 ) {}
foreach $array as $key => $value {}
foreach $array as $value {}
```

```
$arr = array("one", "two", "three");
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}

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

Controlli

```
break
continue
return
include include_once
require require_once

switch ( expr ) {
   case expr:
      break;
   default:
      break;
}
```

Funzioni

```php
function foo ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
```

```php
function foo()
{
  function bar()
  {
    echo "I don't exist until foo() is called.\n";
  }
}

/* We can't call bar() yet since it doesn't exist. */

foo();

/* Now we can call bar(),
   foo()'s processesing has
   made it accessable. */

bar();
```

Nelle funzioni si inserisce qualsiasi codice PHP valido, compreso funzioni e classi.

L'ordine non è importante salvo nel caso di definizioni condizionali

Argomenti delle funzioni

```
function takes_array($input)
{
    echo "$input[0] + $input[1] = ",
                          $input[0]+$input[1];
}
```

```
function add_some_extra(&$string)
{
    $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str;
```



Passaggio by Reference

Argomenti di default

```
function makeyogurt ($flavour, $type = "acidophilus")
{
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");
```

Gli argomenti con default devono essere a destra di tutti gli altri

Argomenti a numero variabile

```
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br>\n";
    if ($numargs >= 2) {
    echo "Second argument is: ".func_get_arg(1)."<br>\n";
    }
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
    echo "Argument $i is: ".$arg_list[$i].   "<br>\n";
    }
}
```

Valori di ritorno

```
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();



function &returns_reference()
{
    return $someref;
}

$newref =& returns_reference();
```

Sempre + difficile...

```
function foo()
{
    echo "In foo()<br>\n";
}

function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br>\n";
}

function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();

$func = 'bar';
$func('test');

$func = 'echoit';
$func('test');
```

Pratica

Scrivere un form a più campi e un metodo POST/GET per la presentazione dei dati inseriri a video i forma testuale o tabellare.

Suggerimento:

```
<form name="formLogin" action="login.php" method="post">
<table cellpadding=6 cellspacing=0 border=1>
        <font size="+1">CONTROLLO DI ACCESSO</font>
        <p>
        </td>
        </tr>
        <tr>
                <td>Nome utente:</td>
                <td><input type="text" name="login" width=20 maxlength=20></td>
        </tr>
        <tr>
                <td>Password:</td>
                <td><input type="password" name="password" width=20 maxlength=20>
                </td>
        </tr>
</table>
        <p>
        <input type="button" value="Entra" onClick="document.formLogin.submit()">
</form>
```

Security

- Problematiche generali sull'uso di PHP come **CGI** program
- Problematiche generali sull'uso di PHP come modulo Apache

Semplice autenticazione

```
if (!isset($_SERVER['PHP_AUTH_USER'])) {
   header('WWW-Authenticate: Basic realm="My Realm"');
   header('HTTP/1.0 401 Unauthorized');
   echo 'Text to send if user hits Cancel button';
   exit;
 } else {
   echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
   echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
 }
```

Un semplice metodo di autenticazione prevede l'uso della autenticazione base del server Apache e può essere usata in molti casi per garantire l'accesso ai soli utenti autorizzati.

La funzione header() deve essere usata prima di ogni altro output al browser!

Cookies

```
$value = 'something from somewhere';

setcookie("TestCookie", $value);
setcookie("TestCookie", $value,time()+3600); /* expire in 1 hour */
setcookie("TestCookie", $value,time()+3600, "/~rasmus/",
          ".example.com", 1);
```

- Come per header() deve essere usata prima di ogni output...
- Per cancellare un cookie, occorre passare un valore “”
- Il valore è disponibile dalla pagina successiva
- Il cookie può anche essere un array e viene reso disponibile come tale nel codice:

```
setcookie ("cookie[three]", "cookiethree");
setcookie ("cookie[two]", "cookietwo");
setcookie ("cookie[one]", "cookieone");
```

… e uso

```
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
        echo "$name : $value <br />\n";
    }
}
```

I cookies vengono generalmente usati per conservare informazioni persistenti di sessione, piuttosto che informazioni di comunicazione da pagina a pagina.

Bisogna inoltre sempre considerare il fatto che le informazioni collegate non devono essere “sensibili”

Sessioni

```php
// page1.php

session_start();

echo 'Welcome to page #1';

$_SESSION['favcolor'] = 'green';
$_SESSION['animal']   = 'cat';
$_SESSION['time']     = time();

// Works if session cookie was accepted
echo '<br /><a href="page2.php">page 2</a>';

// Or maybe pass along the session id, if needed
echo '<br /><a href="page2.php?' . SID . '">page 2</a>';
```

session_start() prima di ogni altro output...

... e uso

```
// page2.php

session_start();

echo 'Welcome to page #2<br />';

echo $_SESSION['favcolor']; // green
echo $_SESSION['animal'];   // cat
echo date('Y m d H:i:s', $_SESSION['time']);

// You may want to use SID here, like we did in page1.php
echo '<br /><a href="page1.php">page 1</a>';
```

sessione_destroy() può essere usato per eliminare una sessione precedentemente creata.

session_unset() può essere usato per eliminare tutte le variabili registrate in $_SESSION.

unset() per eliminare le *singole* variabili di sessione.

MySQL

MySQL è un DBMS **relazionale** che implementa un dialetto e sottoinsieme dello standard **SQL99** come DML e DDL.

Ha una doppia licenza GPL e commerciale e attualmente è alla versione 4.0.22.

Sito di riferimento http://www.mysql.com

? RELAZIONALE ?

***Street man definition*: struttura di dati complessa organizzata in tabelle a campi fissi e relazioni o link fra i dati di queste. Un DBMS è un programma modulare che si incarica della gestione diretta della creazione, ricerca e inserimento di dati delle tabelle e garantisce accesso concorrente e sincronizzato mediante un linguaggio di alto livello (per es. SQL)**

Bignami di SQL

```
drop database if exists Esempio;
create database Esempio;
use Esempio;
drop table if exists Persone;
create table Persone (
        Id                      int unsigned not null auto_increment,
        Nome                    tinytext,
        Cognome                 tinytext,
        Via                     tinytext,
        Citta                   tinytext,
        Provincia               char(2),
        Zip                     tinytext,
        Nazione                 tinytext,
        Telefono                tinytext,
        Cellulare               tinytext,
        Fax                     tinytext,
        Email                   tinytext,
        CodiceFiscale           tinytext,
        DataNascita             date,
        CittaNascita            tinytext,
        ProvinciaNascita        char(2),
        NazioneNascita          tinytext,
        Nota                    text,
        primary key(Id)
);
```

Bignami SQL

```
drop table if exists Societa;
create table Societa (
        Id                      int unsigned not null auto_increment,
        RagioneSociale          tinytext,
        Soundex                 char(4),
        rifTipo                 int unsigned references TipiSocieta,
        rifAmministratore       int unsigned not null references Persone,
        PartitaIva              tinytext,
        CodiceFiscale           tinytext,
        Via                     tinytext,
        Citta                   tinytext,
        Provincia               char(2),
        Nazione                 tinytext,
        Zip                     tinytext,
        Telefono                tinytext,
        Fax                     tinytext,
        Email                   tinytext,
        Nota                    text,
        index(rifAmministratore),
        primary key(Id)
);
```

Bignami SQL

```
drop table if exists TipiSocieta;
create table TipiSocieta (
        Id                      int unsigned not null auto_increment,
        Sigla                   varchar(20),
        Descrizione             tinytext,
        primary key(Id)
);

insert into TipiSocieta values (
                null,
                "S.R.L.",
                "Società a responsabilità limitata"
);
insert into TipiSocieta values (
                null,
                "S.P.A.",
                "Società per azioni"
);
```

Consultazione di una tabella

SELECT

```
SELECT * FROM Persone WHERE Cognome LIKE 'Rossi';

SELECT Nome AS n, Cognome AS c FROM Persone;

SELECT COUNT(*) FROM Societa WHERE  Telefono IS NOT NULL;

SELECT t1.RagioneSociale FROM Societa AS t1 LEFT JOIN Persone AS
t2 on t1.rifAministratore=t2.Id WHERE t2.Cognome LIKE 'Ross%';
```

Inserimenti e Aggiornamenti

INSERT, UPDATE, ALTER, DELETE

```
INSERT INTO Persone (Nome,Cognome) VALUES
('Mario','Rossi');

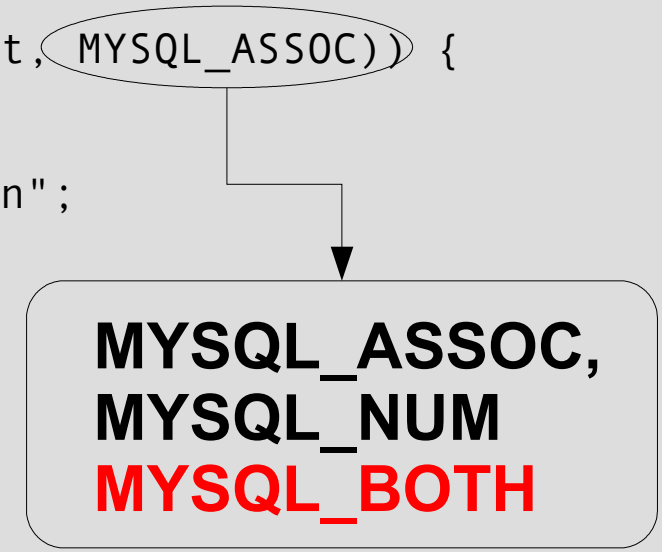
UPDATE Persone SET Cognome='ROSSI' WHERE Id=1;

ALTER TABLE Persone ADD COLUMN Nickname TINYTEXT
AFTER Nota;

DELETE FROM Persone WHERE Id>120;
```

Funzioni principali di PHP

```php
<?php
    /* Connecting, selecting database */
    $link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
        or die("Could not connect : " . mysql_error());
    print "Connected successfully";
    mysql_select_db("my_database") or die("Could not select database");
    /* Performing SQL query */
    $query = "SELECT * FROM my_table";
    $result = mysql_query($query) or die("Query failed : " . mysql_error());
    /* Printing results in HTML */
    print "<table>\n";
    while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
        print "\t<tr>\n";
        foreach ($line as $col_value) {
            print "\t\t<td>$col_value</td>\n";
        }
        print "\t</tr>\n";
    }
    print "</table>\n";
    /* Free resultset */
    mysql_free_result($result);
    /* Closing connection */
    mysql_close($link);
?>
```



Funzioni equivalenti

```
mysql_fetch_row($result)

mysql_fetch_object($result)

mysql_fetch_assoc($result)

mysql_fetch_array($result,MYSQL_BOTH)
```

La scelta è puramente una questione di gusti e opportunità

Funzioni utili

```
string mysql_escape_string ( string unescaped_string )

int mysql_num_rows ( resource result )

int mysql_insert_id ( [resource link_identifier] )

resource mysql_list_dbs ( [resource link_identifier] )

resource mysql_list_tables( string database [, resource
link_identifier])

resource mysql_list_fields ( string database_name, string
table_name [, resource link_identifier])

bool mysql_create_db ( string database_name [, resource
link_identifier])

string mysql_error ( [resource link_identifier])
```

La programmazione ad oggetti “OOP”

1. Il mondo reale è fatto di oggetti
2. Dato che un programma è la rappresentazione nel calcolatore di una realtà di interesse allora anch’esso è composto da oggetti, detti *oggetti software*
3. Il *programma* è la descrizione dettagliata degli oggetti software: con la OOP utilizziamo concetti del mondo reale piuttosto che concetti legati al computer
4. L’esecuzione di un programma consiste nella cooperazione degli di oggetti software definiti dal programma: ogni oggetto ha il suo compito e l'unione dei compiti determina il funzionamento del programma

Classi ed Oggetti (1)

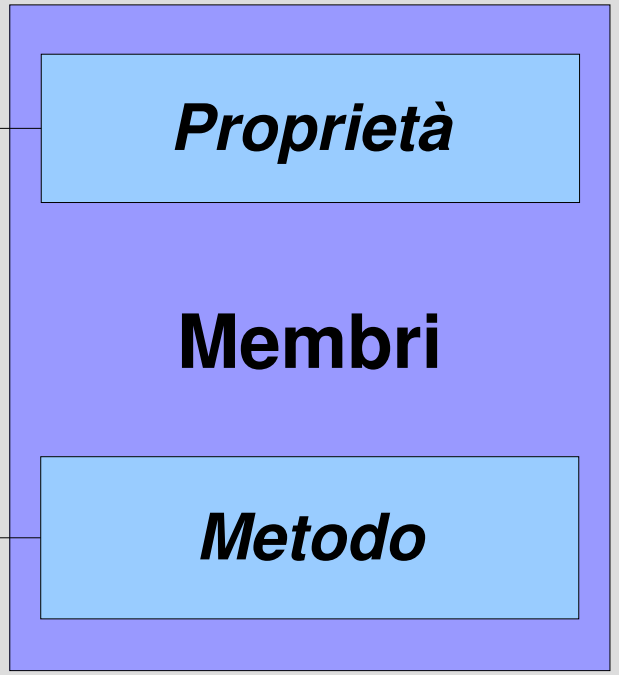
Gli elementi fondamentali della OOP sono le ***CLASSI***:

- una classe definisce un nuovo tipo di dato fornendo le direttive e la forma che un oggetto deve assumere;
- una classe può essere costituita sia da dati che dal codice che opererà su di essi;
- una classe collegherà i dati con il codice;
- una classe è un **CONCETTO ASTRATTO**, essa è un'astrazione logica della realtà che vogliamo rappresentare in un programma.

Classi ed Oggetti (2)

Vediamo un primo esempio di una classe:

```
<?php
    class queue {
        private $data=array();
        function insert($dato){..}
        function get(){..}
        function size(){..}
    }
?>
```



Classi ed Oggetti (3)

I *membri* di una classe sono:

- *Proprietà*, esse sono sostanzialmente le variabili all'interno di una classe e definiranno lo stato di un oggetto;
- *Metodi*, essi agiscono sulle proprietà modificando lo stato dell'oggetto; i metodi sono le funzioni che operano sulle variabili di un oggetto

Classi ed Oggetti (4)

Le classi godono generalmente di due importanti proprietà:

- *ereditarietà* → Riusabilità/ Specializzazione
- *incapsulamento* → Protezione/ Mascheramento

Classi ed Oggetti (5)

Ereditarietà

Nella OOP spesso si ricorre all'uso di una *gerarchia* di classi che svolgono/definiscono compiti via via più specializzati

```
class Persona {
    protected $nome;
    public set_nome($nome){..}
    public get_nome(){..}
}
```

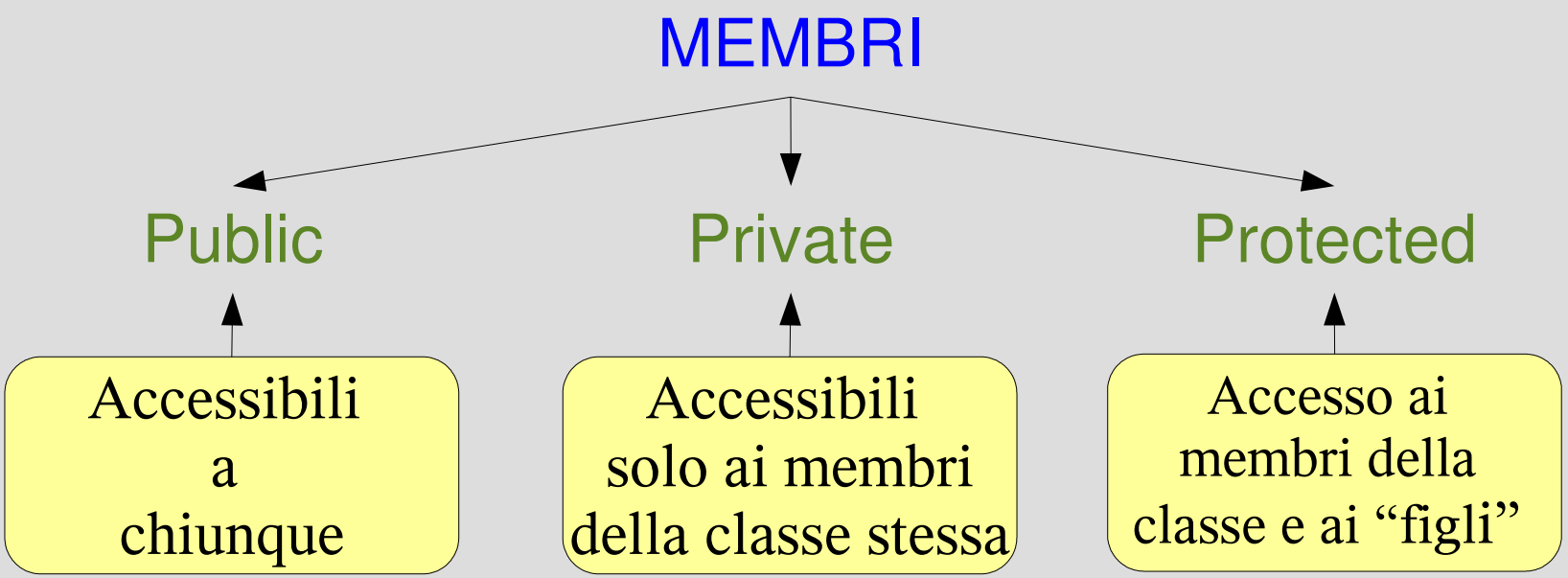
```
class Studente extends Persona {
    private $universita;
    public set_uni($universita){..}
}
```

```
class Impiegato extends Persona {
    private $azienda;
    public set_az($azienda){..}
}
```

Classi ed Oggetti (6)

Incapsulamento

Con *incapsulamento* si intende la possibilità di nascondere i dettagli di implementazione di una classe all'esterno; ciò consente di mostrare solo l'interfaccia della classe



Classi ed Oggetti (7)

Esempio tratto direttamente da www.php.net

```
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}
```

Risultati di alcune azioni

```
echo $obj->public;     // Ok
echo $obj->protected;//  Errore
echo $obj->private;    //  Errore
$obj->printHello();    //  Mostra Tutto
```

Classi ed Oggetti (8)

Dopo aver introdotto il concetto astratto delle *classi,* possiamo chiarire il concetto di *oggetto:*

Un *OGGETTO* *è* un'*istanza* di una classe

Ciò significa che la classe rappresenta la struttura, l'oggetto l'applicazione della stessa.

Es.

```
class Classe {..}
$oggetto = new Classe;
```

Accedere ai membri di una Classe/Oggetto

L'accesso ai membri di una classe/oggetto avviene tramite due operatori:

-> esso permette di accedere a qualsiasi membro di un **oggetto**

:: esso permmette l'accesso diretto ad un **metodo statico** di una **classe**; un metodo si dice **statico** quando non accede a nessuna proprietà della classe

Accedere ai membri di una Classe/Oggetto (2)

```
class MyClass
{
   public $public = 'Public';

   function printPublic()
   {
      echo $this->public;
   }

   function printParam($p)
   {
      echo “<div> $p </div>”;
   }
}
```

```
$obj = new MyClass;
$obj->public = “Ciao”;
$obj->printPublic();
```

Ciao

```
MyClass::printParam(“Auè”);
```

 Auè

Accedere ai membri di una Classe/Oggetto (3)

Alcune annotazioni:

Non è possibile usare una sintassi del tipo

$classA->$classB->function()

dove $classB è dichiarata/istanziata in $classA

Benchè non sia necessario, all'interno di una classe è preferibile accedere ai membri della stessa usando la parola chiave $this

$this->function()

Costruttori

Un **costruttore** è un particolare *metodo* di una classe che serve ad inizializzare le **proprietà** della classe stessa.

In PHP 4 il costruttore ha sempre lo stesso nome della classe che inizializza

```
class Persona {
    var $nome;
    Persona($nome) {
        $this->nome = $nome;
    }
    .
    .
}
```

```
$obj_pers = new Persona("Pippo");
```

Serializzazione di Oggetti

Serializzare un **oggetto** significa convertirlo in un *bytestream* generalmente usato per la memorizzazione in un file. La principale applicazione della serializzazione sono le *sessioni*.

La serializzazione è un processo automatico tuttavia potrebbe essere necessario, prima di serializzare/deserializzare un **oggetto,** compiere alcune operazioni su di esso: esistono due *metodi* predefiniti che ci permettono di fare ciò

__sleep() **__wakeup()**

Serializzazione di Oggetti (2)

__sleep()

questo metodo viene chiamato subito **prima** della **serializzazione** di un oggetto. E' utile per eseguire operazioni di pulitura su un oggetto stesso (es, la chiusura di un file)

__wakeup()

questo metodo viene chiamato subito **dopo** la **deserializzazione** di un oggetto. E' utile per eseguire operazioni di “preset” su un oggetto (es, la chiusura di un file, l'apertura di connessioni ad un database)

Serializzazione di Oggetti (3)

```
Class Log {
    var $filename;
    var $fp;
    .
    .
    __wakeup() {
        $this->fp = fopen($filename,'a');
        .
    }
    .
    __sleep() {
        fclose($this->fp);
        .
    }
}
```

Introspezione

L'***introspezione*** è la capacità di un programma di esaminare le caratteristiche di un oggetto come il suo nome, la sua classe genitore, i suoi metodi.

In PHP esistono delle funzioni/metodi appositamente dedicati allo scopo

- class_exists(*classname*);
- get_declared_classes(); <--- restituisce un array
- get_class_methods(*classname*);
- get_class_vars(*classname*);
- get_parent_class(*classname*);
- is_object(*$var*);
- get_class(*$obj*);

Una precisazione sull'ereditarietà

I membri di una classe figlia con lo stesso nome della classe genitore hanno, all'interno della classe figlia, il “predominio” rispetto a quelli ereditati; tuttavia, in alcuni casi, potrebbe essere necessario usare i metodi dei genitori e semplicemente espanderli: in tal caso ci viene in aiuto la notazione ***parent::metodo_genitore()***

```
class Persona {
    var $nome;
    set_nome($nome){..}
    get_nome(){..}
}
```

```
class Studente extends Persona {
    var $universita;
    set_uni($universita){..}
    set_nome($nome){
        parent::set_nome($nome);
        echo “Nome classe figlia”;
    }
}
```

Una applicazione complessa

L'obbiettivo di questo esercizio è realizzare una semplice applicazione di gestione di un catalogo di libri. L'applicazione deve poter consentire l'inserimento, la modifica, la cancellazione, la ricerca secondo vari criteri (es. per titolo, per autore, per anno, per titolo, per parola chiave).

Deve inoltre consentire l'esportazione e/o stampa in un formato semplice poi importabile in altri programmi il CSV.

Deve consentire l'importazione nello stesso formato e magari in altri.

Non è necessario essere stilisticamente corretti (XHTML), ci si concentri sulla logica di funzionamento.

Da dove si parte?

I dati e le relazioni che li legano sono la base di partenza di almeno il 90% del software. Il codice che manipola direttamente tali dati ne deriva di conseguenza tipicamente in maniera *bottom-up*.
Il resto del programma deriva in maniera *top-down* partendo da quello che il programma deve *proceduralmente* fare.

Alcune scelte architetturali e di disegno della applicazione dovranno comunque essere fatte indipendentemente dalla fase di analisi dati e funzionale

Programmazione per sottosistemi e moduli

I dati

Un libro è caratterizzato minimalmente da:

Titolo, Sottotitolo, Autori, Edizione, Editore, Anno di pubblicazione, Collana, Genere, Codice ISBN, Numero di pagine, Lingua, prezzo, rilegatura, data di acquisto,...

Questo già ci pone una serie di scelte in merito ai dati effettivamente coinvolti, e alla sottostrutturazione di questi.

Editori e Autori dovranno appartenere a dataset differenti?
Come rappresenteremo i singoli campi?
Ci sono chiavi di accesso naturali? Quali sono?
Le scelte fatte sulla strutturazione dei dati influenzeranno direttamente la facilità o complessità di implementazione di specifiche funzioni.