Il linguaggio PHP

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

PHP

- **acronimo ricorsivo per “PHP: Hypertext Preprocessor”**
  - in origine "Personal Home Page"
- **linguaggio di scripting open-source usabile per:**
  - script server-side all’interno di pagine web
  - script a riga di comando (come PERL o Python)
  - scrittura di applicazioni desktop (PHP-GTK), anche se probabilmente non è il linguaggio più adatto
- **multi-piattaforma (hw, OS, web server)**
  - codice altamente portabile
- **http://www.php.net**

Versioni PHP

versione	data creazione	data fine supporto
1.0	1995 / 6	
2.0	1997 / 2	
3.0	1998 / 6	2000 / 10
4.0	2000 / 5	2001 / 6
5.0	2004 / 7	2005 / 9
5.6	2014 / 8	2018 / 12
6.x	mai	N.A.
7.0	2015 / 12	2018 / 12
7.3	**2018 / 12**	**2021 / 12**
7.4	2019 Q4	2022 Q4

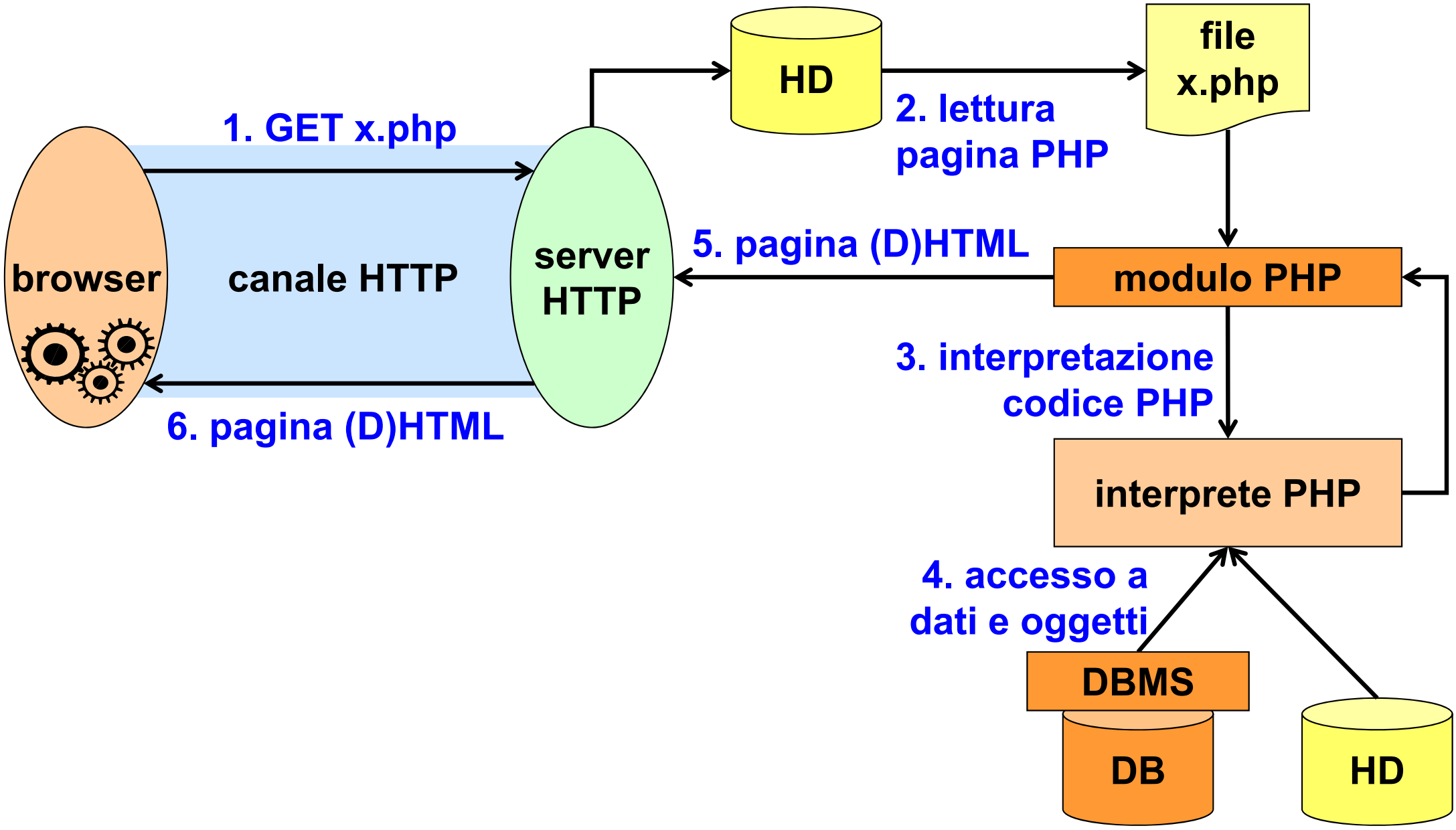
Installazione

- **PHP può essere installato**
  - come modulo del server HTTP
    (disponibile per i maggiori server: Apache, IIS, …)
  - come interprete di CGI
    (sconsigliato se non ci sono ragioni particolari)
- **XAMPP**
  - pacchetto con apache+php+mariaDB
  - per Windows, Mac, Linux
  - facile da installare e gestire
  - www.apachefriends.org

Architettura

- **modulo PHP:**
  - colloquia con il server HTTP tramite l'interfaccia SAPI
  - interpreta i file PHP
- **file PHP:**
  - file di testo con estensione .php (.php3, .phtml)
    - estensione specificata nella configurazione del server HTTP
  - consiste in HTML standard e linguaggio di script racchiuso tra tag speciali (es. "<?php" e "?>")

Architettura



Un primo esempio di pagina PHP

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Saluti</title>
</head>
<body>
  <h1>
  Informazioni sulla versione di PHP installata
  </h1>
  <?php phpinfo(); ?>
</body>
</html>
```

Esempio di pagina PHP

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Saluti</title>
</head>
<body>
  <?php
  for ($i=1; $i<=5; $i++)
     printf ("<h%d>Ciao!</h%d>\n", $i, $i);
  ?>
</body>
</html>
```

Risultato dell'elaborazione (trasmesso al browser)

```
<html>
<head>
  <meta charset="utf-8">
  <title>Saluti</title>
</head>
<body>
<h1>Ciao!</h1>
<h2>Ciao!</h2>
<h3>Ciao!</h3>
<h4>Ciao!</h4>
<h5>Ciao!</h5>
</body>
</html>
```

Il tag PHP

- `<?php ... ?>`
  - sempre disponibile
  - raccomandato per portabilità
- `<? ... ?>`
  - solo su server configurato con "short open tag"
  - frequente ma non garantito (e spesso fuori dal nostro controllo)
- `<script language="php"> ... </script>`
  - (rimosso da PHP7) generale ma poco usato
- `<% ... %>`
  - (rimosso da PHP7) solo su server con "ASP tag"

Caratteristiche del linguaggio

- **sintassi simile a C, con molte varianti**
- **è case-insensitive per certi aspetti (es. nomi funzioni), case-sensitive per altri (es. nomi variabili)**
  - conviene sempre pensarla case-sensitive
- **fine comando delimitato da “ ; ”**
- **commenti:**
  - da "**//**" (oppure "**#**") sino a fine riga
  - racchiusi tra "**/***" e "***/**" (possono occupare più righe)
- **attualmente versioni 3, 4, 5, 6 e 7**
  - abbastanza compatibili per semplici applicazioni
  - con sostanziali differenze

Identificatori

- **caratteristiche generali**
  - contiene caratteri alfanumerici o _
  - inizia con carattere alfabetico o _
  - sono considerati alfabetici i caratteri ASCII 127-255 (uso sconsigliato)
- **esempi:**
  - totale
  - voto2
  - _27
  - años

Variabili

- **identificatore preceduto dal carattere $ (es. $tot)**
- **identificatore case-sensitive**
- **non è necessario dichiararle prima dell'uso**
  - valore di default se non inizializzate (0, 0.0, "")
- **non sono tipate (al contrario di Java, C, …) ma undefined (come in JS)**
  - prendono il tipo al momento dell'inizializzazione
  - conversione automatica di tipo al momento dell'uso
- **funzione isset(*var*) per verificarne l'esistenza**
- **funzione unset(*var*) per distruggere una variabile**
- **$this è una variabile riservata**

Tipi di dati

- **tipi scalari**
  - Booleani (boolean)
  - numeri interi (integer)
  - numeri frazionari (float, double) sempre double
  - stringhe di caratteri (string)
- **tipi compositi**
  - vettori (array)
  - oggetti (object)
- **tipi speciali**
  - resource
  - NULL

Informazioni di tipo

- **var_dump(*expr*)**
  - tipo e valore di un'espressione
  - nella forma *tipo(valore)*
- **gettype(*expr*)**
  - tipo di un'espressione
- **is_int( ), is_float( ), is_double( ), is_string( ), …**
  - vero o falso se l'espressione è del tipo indicato

Numeri interi

- **hanno sempre il segno (non esistono 'unsigned')**
- **esprimibili in varie basi:**
  - decimale (es. 33)
  - esadecimale (es. 0x21)
  - ottale (es. 041)
  - binario (es. 0b100001)
- **in caso di overflow non viene generato errore ma sono trasformati in float automaticamente (!)**
- **la conversione da float a integer è imprevedibile**
  - meglio controllarla esplicitamente tramite una delle seguenti funzioni: round( ), ceil( ) o floor( )

Stringhe

- **rappresentabili in vari modi**
  - prendono tutti i caratteri (incluso il ritorno a capo!) sino al terminatore
- **single-quote**
  - es. `'Ciao John O\'Hara'`
  - non espande le variabili e non usa le sequenze di escape (tranne \\ e \')
- **double-quote**
  - es. `"totale = $tot Euro"`
  - espande le variabili ed usa le sequenze di escape del C (più \$, \\ e \")
  - non usare $var ma {$var …} per variabili complesse

Heredoc e Nowdoc

- **per rappresentare stringhe (ed inizializzare vettori o passare parametri complessi a funzioni)**
- **heredoc**
  - `<<< marker` oppure `<<< "marker"`
  - prende tutto il testo dalla riga successiva a quella col marker (che deve iniziare in colonna 1)
  - il testo è considerato una stringa double-quote
- **nowdoc**
  - `<<< 'marker'`
  - come heredoc ma la stringa è del tipo single-quote
- **il marker NON deve essere indentato, deve essere seguito da ; e NIENTE ALTRO**

Generare output (inserito in HTML)

- funzioni echo( ) o print( ) (sinonimi)
- parentesi non necessarie
- esistono anche printf( ) e affini (es. sprint, sscanf)
- <? echo $a ?> si può abbreviare in <?= $a ?>
  - NB. = attaccato a <?
- operatore "." per concatenare stringhe

```
$numero = 101;
$stringa = "La carica";

echo $stringa." dei ".$numero;
// genera "La carica dei 101"
```

printf ( ) e sprintf ( )

- stessa funzionalità e specifica di formato del linguaggio C
- esempio:

```
$a=7;
$b=3;
printf ("<p>%d + %d = %d</p>\n", $a, $b, $a+$b);
printf ("<p>%d / %d = %.2f</p>\n", $a, $b, $a/$b);
```

```
7 + 3 = 10
7 / 3 = 2.33
```

Parametri posizionali

- possibile usare %numero$formato per indicare uno specifico parametro di printf
- in questo caso si devono numerare tutti i parametri
- utile per ripetizioni o con stringa di formato variabile e non si può/vuole cambiare gli argomenti

```
$format = '%1$d! siamo veramente nell'anno %1$d?'
printf ($format, 2015);
```

```
$format = '%02d/%02d/%4d;
```

◄---------- *data formato italiano*

```
$format = '%2$02d/%1$02d/%3$4d';
```

◄----- *data formato inglese*

```
printf ($format, 31, 1, 2015);
```

Variabili variabili ☺

- permettono di avere variabili con un nome variabile
- si ottengono con **$$** come prefisso

```
$var = "pippo";
$pippo = "pluto";
echo $$var; // restituisce "pluto"
// 1. echo $($var)
// 2. echo $pippo
// 3. echo "pluto"
```

Array

- **in PHP esistono solo ordered-map**
- **una map è un tipo che associa un *valore* ad una *chiave***
- **in PHP il valore (l'indice dell'array) può essere:**
  - un intero (array numerici, anche con indici non adiacenti)
  - una stringa (array associativo)
  - entrambe le cose (array misti)
- **ottimizzate in diversi modi, a seconda dell'utilizzo**
- **esistono diverse funzioni per gestire array**

Gestione array

```
//creazione array indicandone gli elementi
$vett = array(3, 2, 5);
// ossia vett[0]=3, vett[1]=2, vett[2]=5
$figli = array("Ada", "Antonio", "Paolo");
// ossia vett[0]="Ada", vett[1]="Antonio", ...

// aggiunta elemento in posizione max+1
$vett[] = -1; // aggiunge vett[3]=-1

// array associativo (l'indice è una stringa)
$albo = array("nome"=>"Topolino", "anno"=>1949);
// ossia $albo["nome"]="Topolino",$albo["anno"]=1949
```

Conversione array - stringhe

- **string implode (string $glue , array $pieces)**
  - crea una stringa collegando fra loro tutti gli elementi dell'array tramite la stringa specificata da $glue
  - nota: funziona anche se l'array contiene un solo elemento (genera stringa con elemento ma senza glue) o è vuoto (genera stringa vuota)
- **array explode (string $sep, string $pieces, int $limit)**
  - spezza una stringa in pezzi separate da $sep
  - mette i pezzi negli elementi dell'array
  - (opzionale) $limit indica il numero massimo di elementi dell'array

Esempi implode

```php
$student = array("Pautasso", "Giovanni", "24");
// creazione riga di file CSV
$student_csv = implode( "," , $student );
echo $student_csv."\n";
// "Pautasso,Giovanni,24"
```

```php
// singoli elementi
$elements = array("Antonio", "Marco", "Paolo");
// creazione lista puntata HTML
$list = implode( "</li>\n<li>",$elements );
echo "<ul><li>".$list."</li></ul>\n";
// <ul><li>Antonio</li>
// <li>Marco</li>
// <li>Paolo</li></ul>
```

Esempio explode

```
// riga di file CSV
$student = "Pautasso,Giovanni,24";
// estraggo i tre componenti
$student_data = explode( "," , $student );
echo $student_data[0];  // Pautasso
echo $student_data[1];  // Giovanni
echo $student_data[2];  // 24
```

```
// riga di file CSV
$student = "Pautasso,Giovanni,24";
// estraggo solo i primi due componenti
$student_data = explode( "," , $student, 2 );
echo $student_data[0];  // Pautasso
echo $student_data[1];  // Giovanni,24
```

Funzioni per ordinamento di array

- **array con indice numerico (ordinati in base ai valori)**
  - sort( ) = ordine crescente
  - rsort( ) = ordine decrescente
- **array associativi**
  - asort( ) = ordine crescente (in base al valore)
  - ksort( ) = ordine crescente (in base alla chiave)
  - arsort() = ordine decrescente (in base al valore)
  - krsort() = ordine decrescente (in base alla chiave)

Esempio sort / ksort

```
$studenti = array ("Paolo", "Antonio", "Marco");
var_dump($studenti); // 0:Paolo 1:Antonio 2:Marco

sort($studenti);
var_dump($studenti); // 0:Antonio 1:Marco 2:Paolo
```

```
$studenti = array (
  "Paolo"=>6,"Antonio"=>10, "Marco"=>8);
var_dump($studenti); // Paolo:6 Antonio:10 Marco:8

asort($studenti);
var_dump($studenti); // Paolo:6 Marco:8 Antonio:10

ksort($studenti);
var_dump($studenti); // Antonio:10 Marco:8 Paolo:6
```

Variabili predefinite

- **PHP mette a disposizione un vasto numero di variabili (array) predefinite**
- **visibili globalmente, senza dichiarazione esplicita, non devono essere instanziate (superglobali)**
  - $GLOBALS: tiene traccia di tutte le var. globali
  - $_SERVER: variabili definite dal web server
  - $_ENV: variabili d'ambiente
  - $_GET, $_POST: variabili di un form HTML
  - $_REQUEST = $_GET unione $_POST
  - $_COOKIE: cookie
  - $_SESSION: variabili di sessione

$_GET, $_POST, $_REQUEST

- **contengono le informazioni trasmesse da un client tramite un form**
  - inviato con GET ($_GET) o POST ($_POST)
  - inviato in qualunque modo ($_REQUEST)
- **si usano come un normale vettore, con chiave pari al nome del campo del form**

```
// se presente nel form, usa la
// variabile $nome
if (isset($_REQUEST["nome"]))
   echo "Benvenuto ".$_REQUEST["nome"];
else
   echo 'errore nel form: manca $nome';
```

$_GET: esempio

```
<form action="http://a.b.com/x.php" method="GET">
  <input type="text" name="nome">
  <input type="Submit">
</form>
```

x.html

```
GET /x.php?nome=MARA HTTP/1.1
Host: a.b.com
```

HTTP

```
. . .
echo "Ciao ".$_GET["nome"];
```

x.php

Ciao MARA

(browser) http://a.b.com/x.php?nome=MARA

Operatori aritmetici

descrizione	*simbolo*
addizione	+
incremento unitario	++
sottrazione	-
decremento unitario	--
moltiplicazione	*
divisione (floating-point)	/
modulo (resto della divisione intera)	%

Operatori di assegnazione

descrizione	*simbolo*	*esempio*	*equivalenza*
assegnazione	`=`	`$a = 5`	
ass. con somma	`+=`	`$a += 5`	`$a = $a + 5`
ass. con sottrazione	`-=`	`$a -= 5`	`$a = $a - 5`
ass. con prodotto	`*=`	`$a *= 5`	`$a = $a * 5`
ass. con divisione	`/=`	`$a /= 5`	`$a = $a / 5`
ass. con modulo	`%=`	`$a %= 5`	`$a = $a % 5`
ass. con concatenazione	`.=`	`$s .= "!"`	`$s = $s."!"`

Operatori logici

descrizione	*simbolo*
uguaglianza (valore)	`==`
identità (valore e tipo)	`===`
disuguaglianza (valore)	`!=`
non identità (valore e tipo)	`!==`
maggiore di / maggiore o uguale a	`>` `>=`
minore di / minore o uguale a	`<` `<=`
AND logico	`&&` `and`
NOT logico	`!`
OR logico	`||` `or`
EX-OR logico	`xor`

Operatori e valori Booleani

- **valori Booleani TRUE e FALSE (case insensitive)**
- **i seguenti valori sono equivalenti a FALSE:**
  - l'intero 0 ed il float 0.0
  - la stringa vuota "" e la stringa "0"
  - un array con zero elementi
  - il tipo speciale NULL
  - undefined
- **qualunque altro valore è equivalente a TRUE**
- **attenzione quindi ai confronti:**
  - (27 == true) fornisce valore Vero
  - (27 === true) fornisce valore Falso

Type cast

- **anche se spesso non necessario, si può forzare il tipo di un'espressione precedendola con (tipo)**
- **i possibili tipi sono:**
  - bool o boolean
  - int o integer
  - float, double o real
  - string
  - array
  - object
  - unset
- **es. $ns = (string) 5; equivale a $ns = "5";**

Controllo di flusso

- **utili per eseguire un programma in modo non sequenziale**
  - if
  - if/else/elseif
  - while
  - do/while
  - for
  - foreach
  - switch/case
  - include e require

Controllo di flusso if-else

- comandi per modificare il flusso del programma in base a delle condizioni

```
if ( condizione1 ) {
    ... istruzioni
}
elseif ( condizione2 ) {
    ... istruzioni
}
else {
    ... istruzioni
}
```

Esempio if-else

```
if ($temperatura < 0)
  echo "l'acqua e' ghiacciata";
elseif ($temperatura > 100)
  echo "l'acqua e' vapore";
else
  echo "l'acqua e' allo stato liquido";
```

Controllo di flusso switch

- test su tanti possibili valori di un'espressione
- è una forma abbreviata di cascata di "if-else"
- "break" per non continuare col caso successivo
- "default" se non si ricade in nessun caso esplicito

```
switch ( espressione ) {
case valore1: ...  istruzioni
   break;
case valore2: ... istruzioni
   break;
...
default: ... istruzioni
}
```

Esempio switch

```
$dado = rand (1, 6);
switch ($dado)
{
case 1:
case 3:
case 5:
   echo "Numero dispari";
   break;
default:
   echo "Numero pari";
}
```

Controllo di flusso while

- **struttura per ripetere un blocco di istruzioni finché una condizione è e rimane vera**
- **le istruzioni del ciclo possono quindi essere eseguite zero o più volte**

```
while ( condizione )
{
    ... istruzioni
}
```

Esempio di while

```
<h1>Tabellina del 7</h1>

<?php
$x = 1;
while ($x <= 10)
{
   echo "<p>7 * $x = ".($x*7)."</p>";
   $x++;
}
?>
```

Controllo di flusso do-while

- **struttura simile al while con la differenza che il controllo si fa alla fine del ciclo e quindi il ciclo viene sempre eseguito almeno una volta**

```
do
{
    ... istruzioni
} while ( condizione );
```

Esempio di do-while

```
$x = 0;
do {
   echo "$x<br />";
   $x++;
} while ($x != 10);
```

Controllo di flusso for

- **struttura per ripetere blocchi di istruzioni finché una condizione rimane vera (simile al while)**
- **specifica:**
  - un'azione di inizializzazione
  - una condizione
  - un'azione da ripetere alla fine di ogni ciclo (tipicamente un incremento/decremento dell'indice associato al ciclo)

```
for ( inizializzazione; condizione; azione_ripetitiva )
{
    ... istruzioni_da_ripetere
}
```

Esempio di for

```
// calcolo della somma
// dei primi 10 numeri naturali

$totale = 0;
for ($i = 1; $i <= 10; $i++) {
  $totale += $i;
}
echo "Somma dei numeri [1...10] = $totale";
```

Controllo di flusso foreach

- **disponibile da PHP 4**
- **serve per scandire tutti gli elementi di un vettore (senza conoscerne la dimensione)**
  - prendendo solo il valore (ma chiave ignota)
  - oppure le coppie chiave (indice) e valore

```
foreach ($vettore as $valore) {
   istruzioni (che operano su $valore);
};

foreach ($vettore as $chiave => $valore) {
   istruzioni (che operamo su $chiave/$valore);
}
```

Esempio di foreach

```
$vettore = array();
$vocab = array (
  "giallo"=>"yellow",
  "rosso"=>"red",
  "verde"=>"green");

for ($i=1; $i<=10; $i++)
  $vettore[$i] = "test".$i;

foreach ($vettore as $val)
  echo "<p>$val</p>";

foreach ($vocab as $key => $val)
  printf ("<p>(IT)%s = (EN)%s</p>\n", $key, $val);
```

count( )

- **numero di elementi in una variabile complessa**
  - array
    - secondo parametro opzionale COUNT_RECURSIVE
    - utile per array multidimensionali (es. matrici)
  - object (solo se implementa l'interfaccia Countable)
- **utile per iterare sugli array con indice numerico**
- **esempio:**

```
echo "<p>Dimensione: ".count($vettore)."</p>";
echo "<p>Elementi: <ul>";
for ($i=0; $i<count($vettore); $i++)
  echo "<li>[$i] = {$vettore[$i]}</li>";
echo "</ul></p>";
```

reset, end, next, prev, key, current, each

- **funzioni per manipolare l'indice implicito di un array associativo e leggerne i corrispondenti valori**
  - indice implicito rappresentato dal nome dell'array
- **reset( ) = si posiziona all'inizio (primo elemento)**
- **end( ) = si posiziona alla fine (ultimo elemento)**
- **next( ) = passa al prossimo elemento**
- **prev( ) = passa al precedente elemento**
- **key( ) = indice dell'elemento corrente (o NULL)**
- **current( ) = valore dell'elemento corrente**
- **each( ) = restituisce chiave e valore e quindi passa al prossimo elemento (restituisce FALSE alla fine) – *deprecata da PHP 7.2***

Esempi di iterazioni su array associativo

```
$vocab = array (
   "giallo"=>"yellow",
   "rosso"=>"red",
   "verde"=>"green");

// elenca in ordine crescente
for (reset($vocab); $k=key($vocab); next($vocab)) {
   $val = $vocab[$k];
   printf ("<p>(IT)%s = (EN)%s</p>\n", $k, $val);
}
// elenca in ordine decrescente
for (end($vocab); $k=key($vocab); prev($vocab)) {
   $val = $vocab[$k];
   printf ("<p>(IT)%s = (EN)%s</p>\n", $k, $val);
}
```

La funzione trim( )

- agisce su stringhe
- restituisce la stessa stringa dopo aver eliminato eventuali spazi iniziali e finali
- utile applicarla su tutti i dati di un form per ripulire i dati prima di validarli e/o usarli (es. spazi non sono validi nei dati numerici)
- esempio:

```
$base = " Evviva ";
echo $base."!!!";
echo trim($base)."!!!";
```

```
 Evviva !!!
Evviva!!!
```

La funzione empty( )

- **agisce su variabili o espressioni**
- **restituisce TRUE se:**
  - "" (stringa vuota), "0" (stringa)
  - 0 (intero), 0.0 (float)
  - NULL, FALSE
  - $array (array senza elementi)
  - $var (una variabile dichiarata ma senza valore)
  - variabile inesistente

Filtraggio dei dati

- **estensione con due scopi:**
  - validazione = verificare se un dato è valido per una certa categoria logica
    - es. FILTER_VALIDATE_EMAIL
  - pulizia (sanitization) = eliminare dal dato i caratteri inappropriati per una certa categoria logica
    - es. FILTER_SANITIZE_EMAIL
    - attenzione! toglie i caratteri invalidi ma NON effettua la validazione del risultato
- **standard da PHP 5.2.0 (prima PECL sperimentale)**

Filtri di validazione (I)

- **FILTER_VALIDATE_...**
- **BOOLEAN**
  - TRUE per "1", "true", "on", "yes" altrimenti FALSE
  - FILTER_NULL_ON_FAILURE
- **EMAIL**
- **FLOAT**
  - opzioni: decimal
  - FILTER_FLAG_ALLOW_THOUSAND
- **INT**
  - opzioni: min_range, max_range
  - FILTER_FLAG_ALLOW_OCTAL / _HEX

Filtri di validazione (II)

- **IP**
  - FILTER_FLAG_IPV4 / _IPV6 / _NO_PRIV_RANGE / _NO_RES_RANGE
- **REGEXP**
  - usa una regular expression Perl
- **URL**
  - solo caratteri ASCII nell'host (non caratteri i18n)
  - FILTER_FLAG_PATH_REQUIRED / FILTER_FLAG_QUERY_REQUIRED
- **note:**
  - (PHP < 5.4.11) +0 e –0 non validi come interi ma come float
  - (PHP >= 7.4.0) min_range e max_range per float

Funzioni di filtraggio

- **bool filter_has_var (int *type*, string *var_name*)**
  - esiste la variabile nella categoria indicata?
  - INPUT_GET / _POST / _COOKIE / _SERVER / _ENV
- **filter_id() = ID del filtro specificato**
- **filter_input() = filtra una specifica variabile di input**
- **filter_input_array() = filtra un array di input**
- **filter_list() = elenca tutti i filtri disponibili**
- **mixed filter_var (*var_name*, *filter*, *options_flags*)**
  - filtra una variabile
- **filter_var_array() = filtra un array**

Esempio di filtraggio (e-mail)

```
if ( ! filter_has_var(INPUT_GET,"mailaddr") )
  echo "<p>Errore: manca indirizzo e-mail</p>";
else {
  $m = trim ($_GET["mailaddr"]);
  if ( ! filter_var($m,FILTER_VALIDATE_EMAIL) )
    echo "<p>Errore: e-mail '".$m."'errato</p>";
  else
    echo "<p>OK, e-mail = ".$m."</p>";
}
```

Esempio di filtraggio (interi decimali)

```
if ( ! filter_has_var(INPUT_GET,"anni"))
  echo "<p>Errore: manca numero anni</p>";
else {
  $a = trim ($_GET("anni"));
  $opt = array("options" =>
    array("min_range"=>18);
  if ( ! filter_var($a,FILTER_VALIDATE_INT,$opt))
    echo "<p>Errore negli anni (o minorenne)</p>";
  else
    echo "<p>OK, anni = ".$a."</p>";
}
```

Esempio di filtraggio (interi dec & hex)

```
if ( ! filter_has_var(INPUT_GET,"anni"))
  echo "<p>Errore: manca numero anni</p>";
else {
  $a = trim ($_GET("anni"));
  $opt = array("options" =>
    array("min_range"=>18, "max_range"=>150),
    "flags" => FILTER_FLAG_ALLOW_HEX);
  if ( ! filter_var($a,FILTER_VALIDATE_INT,$opt))
    echo "<p>Errore negli anni (o minorenne)</p>";
  else
    echo "<p>OK, anni = ".$a."</p>";
}
```

Espressioni regolari in PHP

- **http://www.php.net/manual/en/book.pcre.php**
- **estensione PCRE (Perl-Compatible Regular Expression)**
  - sempre presente da PHP 5.3
  - simili (ma non identiche!) a quelle di Perl
- **racchiuse tra due delimitatori a scelta:**
  - / ... / + ... + # ... # % ... %
  - { ... } < ... >
- **eventuali modificatori dopo ultimo delimitatore**
  - es. match case-insensitive: #Ciao#i

Sintassi espressioni regolari

- **vale quanto già visto in JS (più molto altro)**
- **classi di caratteri**
  - [...] [^ ...]
  - \d \D \s \S \w \W (più altre, es. \b = word boundary)
- **alternative**
  - (... | ...)
- **match particolari**
  - . ^ $
- **ripetizioni**
  - * + ?
  - { N } { Nmin, } { Nmin,Nmax }

Classi di caratteri Posix

- **notazione [:*classe*:]**
- **alpha, alnum, digit, lower, upper, xdigit**
  - alfabetici, alfanumerici, cifre, minuscole, maiuscole, esadecimali
- **ascii (caratteri 0-127)**
- **blank, cntrl, graph , print, punct**
  - spazio o tab, controlli, "grafici", stampabili (=grafici + spazi), segni di interpunzione
- **space, word**
  - \s + VT, \w

Funzioni per espressioni regolari PHP

- preg_filter = regex search and replace
- preg_grep = return array entries matching the pattern
- preg_last_error = error code of the last regex exec
- preg_match_all = global regex match
- preg_match = regex match
- preg_quote = quote regex characters
- preg_replace_callback = regex search and replace using a callback
- preg_replace = regex search and replace
- preg_split = split string by a regex

La funzione preg_match( )

- **int preg_match (string *pattern* , string *subject* , array &*matches* , int *flags* , int *offset*)**
- **verifica se *subject* fa match con *pattern***
- ***matches* = array ptr ove catturare le parti (totale ed eventuali subpattern)**
- ***flags* = PREG_OFFSET_CAPTURE (cattura anche gli offset nell'array *matches*)**
- ***offset* = byte da cui iniziare (default: 0)**
- **restituisce 1 se c'è match, 0 se non c'è, FALSE in caso di errore**
  - attenzione a distinguere FALSE da 0
    - fare test con ===FALSE

Esempi: match semplice e con estrazione

```
if ( preg_match('/^[[:alpha:]!?]{6,12}$/',$pwd) )
  echo "<p>Password OK</p>";
else
 echo "<p>Password non rispetta le regole</p>";
```

```
if ( preg_match('/^(\d{1,2})-(\d{1,2})-(\d{4})$/',
  "13-5-2013", $results) ) {
  printf ("<p>Data %s OK</p>\n", $results[0]);
  printf ("giorno=%s, mese=%s, anno=%s\n",
    $results[1], $results[2], $results[3]);
}
else
 echo "<p>Data con formato errato</p>";
```

include e require

- **quattro funzioni:**
  include, require, include_once, require_once
- **inseriscono un file (passato come parametro) e lo valutano tramite l'interprete PHP**
  - file indicato con path assoluto o relativo (preferibile)
  - solo HTML o PHP
- **in caso di errore:**
  - include* restituisce un warning (parti opzionali)
  - require* restituisce un fatal_error (parti vitali)
- ***_once verifica che il file da inserire non sia già stato inserito, altrimenti ignora il comando.**
  - utile per evitare di ridefinire funzioni (vietato in PHP)

virtual( )

- **disponibile solo su Apache**
- **inserisce un file (passato come parametro) e lo valuta tramite Apache**
  - file indicato come la parte locale della URI (ossia dalla radice del server web)
  - permette non solo HTML e PHP ma anche CGI e SSI, ma in questi caso la risorsa deve essere riconosciuta come peculiare da Apache
    - es. CGI in /cgi-bin/
    - es. SSI in file .shtml

Esempio di include/require

```
require("template_top.php");
include_once("controlli.php");
...
require("template_bottom.php");
```

Funzioni

- le informazioni passate alle funzioni si chiamano parametri
- i parametri vengono specificati tra parentesi dopo il nome della funzione
- il nome delle funzioni è case-insensitive
- le funzioni hanno sempre visibilità globale

```
function nome_funzione ($par1, $par2, ...)
{
   ... istruzioni ...
   return( );
}
```

Esempi di funzioni

```
function somma ($a, $b) { return($a+$b); }
...
echo somma(1,2);
```

```
function minoreDi ($a, $b) {
  if ($a<$b) return (true) else return (false);
}

$a=1;
$b=2;
if (!minoreDi($a,$b))
  echo $a." non e' minore di ".$b;
```

Funzione header( )

- aggiunge un header HTTP (sintassi!)
- header() deve essere richiamato prima di qualsiasi output al browser (come prima riga …)

```
<?
   header("Location: http://www.php.net/");
   // Ridireziona il browser al sito di PHP
   exit;
```

```
<?
   header("Content-type: application/pdf");
   header("Content-Disposition: attachment;
                              filename=downloaded.pdf");

/* ... manda in output un file pdf ... */
```

Visibilità (scope) delle variabili

- di default le variabili sono locali
- le variabili globali vanno dichiarate esplicitamente in ogni funzione che le vuole usare

```
$n = 3;

function somma_ad_n ($x)
{
   global $n;
   return $n + $x;
}
```

Visibilità (scope) delle variabili

- attenzione a global!

```
$n = 3;

function somma_ad_n_errato ($x) {
   return $n + $x;
   // ritorna $x perche' localmente $n=0
}

function somma_ad_n_ver2 ($x) {
   return $GLOBALS["n"] + $x;
}
```

Registrazione delle variabili superglobali

- **fino a PHP 4.2 di default erano registrate**
  - $_GET[“pippo”] accessibile come $pippo
- **da PHP 4.2 di default non sono registrate, ma è possibile modificare il file di configurazione**
  - register_globals = true;
- **dal punto di vista della sicurezza non è bene**
- **la registrazione avviene secondo una scala di priorità (configurabile)**
  - default = EGPCS, env/get/post/cookie/session
  - attenzione alle sovrascritture!
- **meglio evitare le registrazioni!**
- **nota: funzionalità deprecata da 5.2 e rimossa da 5.4**

$_SERVER

- estrae i valori delle variabili dell'intestazione del protocollo HTTP
- la funzione phpinfo( ) mostra tutte le variabili definite, compreso $_SERVER
- i seguenti esempi restituiscono il modello del browser ed il nome DNS del server (come scritto nella URI)

```
<? $client = $_SERVER["HTTP_USER_AGENT"]; ?>
```

```
<? $server = $_SERVER["HTTP_HOST"]; ?>
```

$_SERVER: esempio

```
<table border=1>
<tr>
   <td><i>server variable</i></td>
   <td><i>value</i></td>
</tr>
<?php foreach ( $_SERVER as $key => $value ) {
echo "<tr>
   <td>$key</td>
   <td>$value</td>
</tr>";
} // end foreach ?>
</table>
```

Gestione dei cookie

- meccanismo per memorizzare dati nel browser remoto e tenere traccia degli utenti o identificarli al loro ritorno
- PHP supporta i cookie in maniera trasparente
- $_COOKIE contiene i cookie inviati dal browser
- setcookie( ) e setrawcookie( ) creano un cookie coi parametri indicati

- N.B. l'impostazione dei cookie deve essere fatta prima di qualsiasi output altrimenti genera errore

Funzione setcookie( )

- **restituisce Boolean (TRUE se OK, altrimenti FALSE)**
- **tutti i parametri sono opzionali tranne "name"**
- **per saltare un parametro usare la stringa vuota o l'intero zero (a seconda del tipo)**
- **parametri (nell'ordine):**
  - string *name*, string *value* = nome e valore del cookie
  - int *expire* = scadenza (secondi da Unix Epoch)
  - string *path* = path virtuale della risorsa sul server
  - string *domain* = dominio del cookie
  - bool *secure* = trasmissibile solo in modo sicuro
  - bool *httponly* = usabile solo da HTTP

Funzioni time( ) e mktime( )

- **utili per impostare la data di scadenza dei cookie visto che PHP usa la Unix Epoch:**
  - 1/1/1970 00:00:00 GMT
- **int time( )**
  - secondi da Unix Epoch all'istante attuale
- **int mktime (*ora,minuti,secondi, mese,giorno,anno)***
  - secondi da Unix Epoch alla data indicata
  - si possono omettere parametri da destra a sinistra e saranno considerati come i valori attuali

Esempio impostazione cookie

```
$nome = $_REQUEST("yourname");
$cognome = $_REQUEST("yourfamilyname");
$scadenza = time() + 3600*24; // scade dopo 24 ore

setcookie ("myname", $nome, $scadenza,
  "/",              // qualunque path
  "polito.it"); // per tutti i server di polito.it

setcookie ("mysurname", $cognome, $scadenza,
  "/",              // qualunque path
  "polito.it"); // per tutti i server di polito.it
```

Esempio impostazione cookie (con array e mktime)

```
$nome = $_REQUEST["yourname"];
$cognome = $_REQUEST["yourfamilyname"];
$scadenza = mktime (23, 59, 59, 12, 31, 2023);

setcookie ("myself[name]", $nome, $scadenza,
  "/",              // qualunque path
  "polito.it"); // per tutti i server di polito.it

setcookie ("myself[surname]", $cognome, $scadenza,
  "/",              // qualunque path
  "polito.it"); // per tutti i server di polito.it
```

Esempi lettura cookie

```
<?php if (isset($_COOKIE["myname"]))
   echo "<p>Ciao ".$_COOKIE["myname"]."!</p>" ?>
```

```
<table border=1>
<tr>
  <td><i>cookie name</i></td>
  <td><i>cookie value</i></td>
</tr>
<?php foreach ( $_COOKIE as $key => $value ) { ?>
<tr>
  <td><?= $key ?></td>
  <td><?= $value ?></td>
</tr>
<? } // end foreach ?>
</table>
```

Gestione delle sessioni: $_SESSION

- **disponibile da PHP 4**
- **mantiene informazioni sulla sessione attiva di un client, ad esempio:**
  - il nome utente dopo un login
  - le preferenze (lingua di visita o dimensione font grosso) di un utente anonimo
- **tiene traccia dei movimenti di un client**
- **informazioni associate ad un cookie volatile, in maniera trasparente per il programmatore**
  - nome di default PHPSESSID
- **se i cookie non sono disponibili, parametri passati in GET (pericolo!)**

Funzioni per gestire le sessioni (I)

- **int session_status (void)**
  - possibili risultati
    - PHP_SESSION_DISABLED
    - PHP_SESSION_NONE nessuna sessione attiva
    - PHP_SESSION_ACTIVE c'è una sessione attiva
- **bool session_start(void)**
  - deve essere richiamata **prima** di qualsiasi output
    - perché manipola il cookie nell'header
  - crea una nuova sessione (o usa quella esistente)

Funzioni per gestire le sessioni (I)

- **string session_name ([string *newname*])**
  - restituisce (o imposta) il nome del cookie associato alla sessione (default: PHPSESSID)
  - da richiamare **prima** di session_start()

Funzioni per gestire le sessioni (II)

- **$_SESSION[" *var* "] = *valore***
  - registra la variabile $var nella sessione corrente e le assegna il valore indicato
- **unset($_SESSION["var"])**
  - cancella la variabile $var dalla sessione corrente
- **void session_write_close(void)**
  - salva i dati della sessione e la chiude
  - fatto automaticamente al termine dello script
  - c'è un lock sui dati della sessione finché non si chiama questa funzione (problema con accesso concorrente, ad esempio tramite vari frame)

Esempio: contatore pagine visitate

```
// da mettere in ciascuna pagina
if (session_status() !== PHP_SESSION_ACTIVE)
   session_start();
if (!isset($_SESSION['npag'])) {
   $_SESSION['npag'] = 1;
} else {
   $_SESSION['npag']++;
}

...

printf ("<p>Hai sinora visitato %s pagine.</p>",
   $_SESSION['npag']);
```

Funzioni per gestire le sessioni (III)

- **bool session_destroy(void)**
  - cancella i dati associati alla sessione
  - … ma non l'ID
  - … e neanche le variabili in $_SESSION
- **string session_id(void)**
  - restituisce l'identificativo della sessione (se esiste) altrimenti la stringa vuota
  - identificativo disponibile anche come SID
- **string session_id(string *id*)**
  - fissa l'identificativo della sessione
  - da chiamarsi prima di session_start( )

Funzioni per gestire le sessioni (IV)

- **bool session_regenerate_id (**
  [bool *delete_old_session*])
  - crea un nuovo identificativo della sessione
  - opzionalmente cancella i dati associati alla vecchia sessione (default = false)
- **int session_cache_expire (int time)**
  - specifica un timeout in minuti (default 180')
  - un valore piccolo (es. minore di 4') fa perdere lo stato, troppo grande sovraccarica il server
  - da chiamare **prima** di avviare la sessione

Esempio: cancellazione di una sessione

```
// cancella le variabili della sessione
$_SESSION = array();

// cancella il cookie
$params = session_get_cookie_params();
setcookie(session_name(), '', time()-42000,
   $params["path"], $params["domain"],
   $params["secure"], $params["httponly"]
);

// cancella la sessione da disco
session_destroy();
```

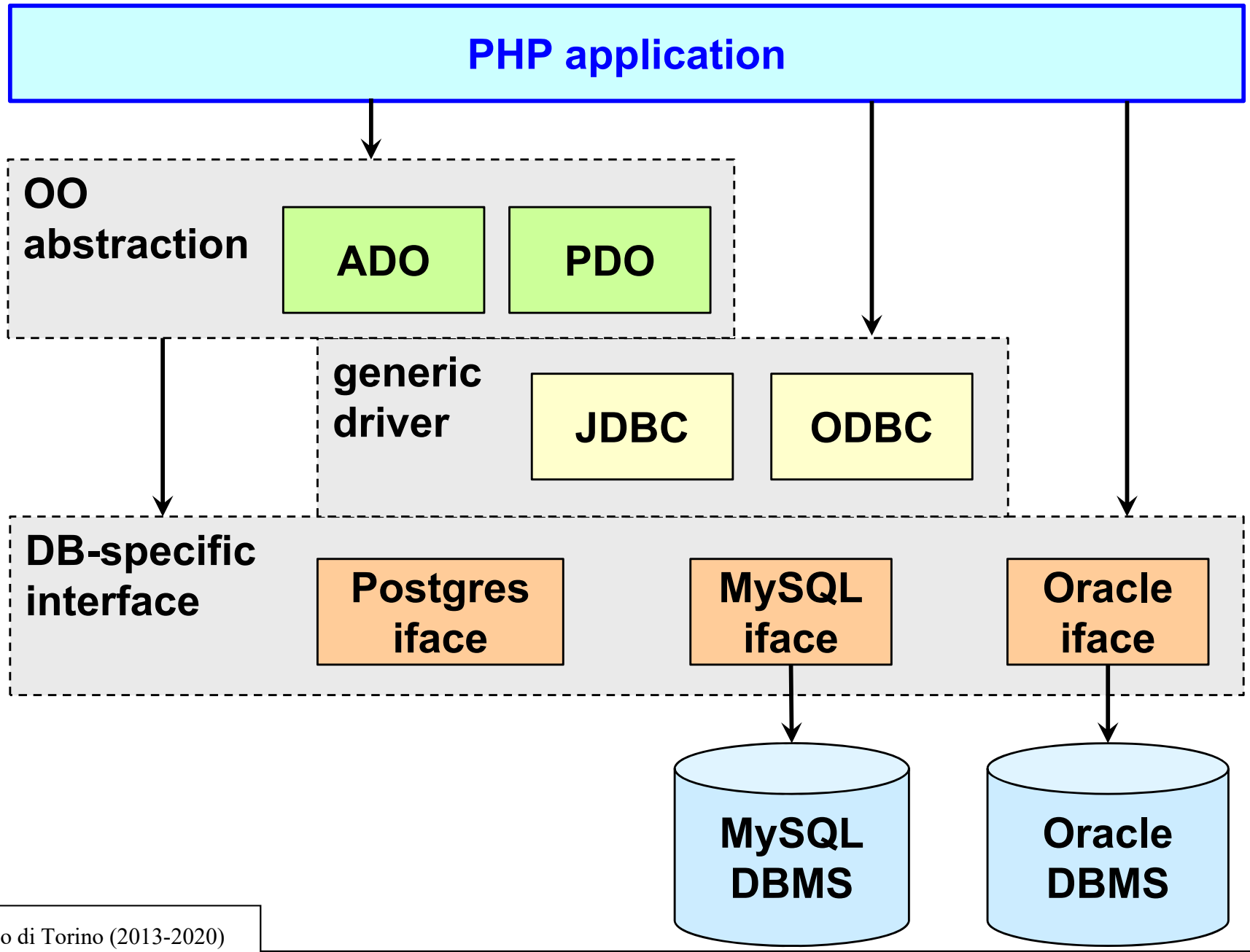
Esempio: timeout manuale delle sessioni

```
MAX_IDLE_T = 1800; // 30 minuti
if (!isset($_SESSION['idle_time']))
  $_SESSION['idle_time'] = time() + MAX_IDLE_T;
else
{
  if ($_SESSION['idle_time'] < time())
  {
    // cancella dati vecchia sessione + crea nuova
    session_regenerate_id (TRUE);
  }
  $_SESSION['idle_time'] = time() + MAX_IDLE_T;
}
```

PHP e database

- **PHP supporta molti DBMS:**
  - dbm, dBase, FrontBase, filePro, Informix, Interbase, Ingres II, Microsoft SQL Server, mSQL, MySQL, ODB, Oracle 8, Ovrimos, PostgreSQL, SESAM, SQLite, Sybase
- **per ciascun DBMS esiste un'API dedicata (simile ma non identica per tutti i DBMS)**
- **esistono anche librerie di alto livello (es. PEAR DB, PDO) per accedere in modo uniforme a qualunque DBMS**

Collegamento PHP-DBMS



Terminologia

- **connessione**
  - canale tra l'interprete PHP ed uno specifico DBMS
- **query**
  - comando SQL emesso dall'interprete PHP verso il DBMS tramite una connessione aperta
- **result set**
  - una tabella (virtuale) coi risultati di una query
  - un result set NON è trasferito automaticamente all'interprete PHP e non c'è garanzia che sia memorizzato nel DB (ossia è volatile)
- **riga (row)**
  - la componente elementare di un result set

Interfaccia MySQLi

- **versione migliorata (i = improved) dell'interfaccia MySQL di basso livello**
- **accesso diretto a SQL**
- **doppia interfaccia:**
  - procedurale
  - ad oggetti
- **supporto dei prepared statement**
  - molto importanti per prestazioni e sicurezza

Connessione al DBMS

- si crea con la funzione

```
$con = mysqli_connect( server, username,
    password, database, port, socket )
```

- restituisce un oggetto che rappresenta la connessione al DBMS (o NULL in caso di errore)
- il numero associato all'errore (zero se no errori):

```
int mysqli_connect_errno( void )
```

- la descrizione testuale dell'errore:

```
string mysqli_connect_error( void )
```

- chiusura della connessione:

```
boolean mysqli_close( con )
```

Esempio: apertura e chiusura connessione

```
$con = mysqli_connect(
   "dbserver.polito.it","lioy","123","studenti");
if ( mysqli_connect_errno() )
   printf ("<p>errore - collegamento al DB
impossibile: %s</p>\n", mysqli_connect_error());
else
{
   // operazioni sul DB
   ...
   // rilascio della connessione al DB
 if (!mysqli_close($con));
     printf ("<p>errore di chiusura connessione
- impossibile rilasciare le risorse</p>\n");
}
```

Organizzazione di una query

- **ciascuna query è composta di 4 fasi (3 + 1):**
  - preparazione = costruzione di una stringa contenente il comando SQL da eseguire
  - esecuzione = invio del comando SQL al DBMS, generazione e ricezione del result set
  - estrazione = lettura dei dati presenti nel result set
  - formattazione = costruzione del codice HTML per visualizzare i risultati nella pagina
- **estrazione e formattazione solitamente sono organizzate in un ciclo (while o for)**
- **la formattazione è opzionale, dipende dalle esigenze (presentazione o uso per altre funzioni)**

Query: preparazione

- **comando SQL fisso**
  - esempi

```
$query = "SELECT login FROM utenti";
$query = "INSERT INTO utenti (login,
password) VALUES ('pippo', 'xyz')";
```

- **comando SQL con valori variabili**
  - esempio:

```
$query = "SELECT login FROM utenti
WHERE password = '" . $password . "'" ;
```

  - si usa l'operatore di concatenazione (.) per inserire variabili PHP nella query SQL (N.B. usare gli apici singoli '…' intorno ad ogni stringa)

Query: esecuzione

- una volta preparata, è sufficiente inviare la query SQL al DBMS, usando la connessione attiva ed il database attualmente selezionato:

```
$result = mysqli_query($con, $query)
```

- per le query di tipo INSERT, DELETE o UPDATE il lavoro è terminato
- per le query di tipo SELECT occorre estrarre ed analizzare il result set
- il valore di ritorno è un "riferimento" al result set attraverso cui si arriva al risultato vero e proprio
- liberare la memoria una volta esaminato il risultato

```
mysqli_free_result($result)
```

Esempio: esecuzione query

```
$result = mysqli_query($con,"SELECT * FROM 01NBE")
if (! $result)
   printf ("<p>errore - query fallita: %s<p>\n",
      mysqli_error($con));
else
{
   // operazioni sul result set
   ...
   // rilascio della memoria associata al res.set
   mysqli_free_result($result);
}
```

Query: estrazione

- **si basa sulla funzione**

  ```
  array mysqli_fetch_assoc( rset )
  ```

- **`rset` è il result set ottenuto da `mysqli_query`**
- **il risultato è un array associativo**
  - **indici = nomi dei campi**
- **ad ogni chiamata restituisce:**
  - un array coi valori della prossima riga del result set
  - oppure NULL quando è terminato il result set
- **numero di elementi nel result set dato da**

  ```
  int mysqli_num_rows( rset )
  ```

Esempio: estrazione (e formattazione)

```
... // apertura connessione

... // esecuzione query

while ($row = mysqli_fetch_assoc($result))
{
   printf("<p>ID:%s, cognome: %s, nome: %s</p>\n",
      $row["id"], $row["surname"], $row["name"]);
}

... // chiusura connessione
```

Esempio: estrazione (e formattazione)

```
... // apertura connessione + esecuzione query

$nrow = mysqli_num_rows($result);
printf("<p>Trovati %d record:\n<ul>\n", $nrow);

for (i=1, i<=$nrow; i++)
{
   $row = mysqli_fetch_assoc($result);
   printf(
      "<li>record %d = ID:%s, cognome: %s</li>\n",
      $i, $row["id"], $row["surname"]);
}
echo "</ul>\n"

... // chiusura connessione
```

Prepared statement (I)

- **sono query SQL con struttura fissa ma dati variabili (di input e/o di output)**
- **MOLTO utile per prevenire attacchi SQL injection**
- **creazione di un prepared statement:**

  `stmt = mysqli_prepare( conn, query* )`

  - query* può contenere `?` per i dati variabili
- **dati associati a variabili tramite:**

  `mysqli_stmt_bind_param(stmt,types,vars)`

  - vars è l'elenco delle variabili da associare e types è una stringa che ne specifica ordinatamente il tipo:
    - i = integer, d = double, s = string

Prepared statement (II)

- esecuzione di un prepared statement:

```
mysqli_stmt_execute( stmt )
```

- associazione del risultato (tutti i campi!) a variabili:

```
mysqli_stmt_bind_result( stmt, vars )
```

- caricamento dei valori del risultato nelle variabili:

```
mysqli_stmt_fetch( stmt )
```

- termine uso di un prepared statement:

```
mysqli_stmt_close( stmt )
```

- funzioni per gestione errori:

```
int mysqli_stmt_errno( statement )
string mysqli_stmt_error( statement )
```

Esempio: prepared statement

```
$stmt = mysqli_prepare(
   $con, "SELECT surname FROM 01NBE WHERE id=?");

mysqli_stmt_bind_param($stmt, "i", $matricola);

$studenti = array (12345, 11223, 54321);
foreach ($studenti as $matricola)
{
   mysqli_stmt_execute($stmt);
   mysqli_stmt_bind_result($stmt, $cognome);
   mysqli_stmt_fetch($stmt);
   printf("matr. %d = %s\n", $matricola, $cognome);
}
mysqli_stmt_close($stmt);
```