

Introduzione a PHP 5

Corso Interazione Uomo – Macchina
AA 2005/2006

Un po' di storia 1/2

- A metà degli anni Novanta il Web era ancora formato in gran parte da pagine statiche, cioè da documenti HTML il cui contenuto non poteva cambiare fino a quando qualcuno non interveniva manualmente a modificarlo
- PHP nasce nel 1994, ad opera di Rasmus Lerdorf, come una serie di macro la cui funzione era quella di facilitare ai programmatori l'amministrazione delle home page personali: da qui trae origine il suo nome, che allora significava appunto **Personal Home Page**
- Essendo un progetto di tipo open source, ben presto si formò una ricca comunità di sviluppatori che portò alla creazione di PHP3
- Alla fine del 1998 erano circa 250.000 i server Web che supportavano PHP: un anno dopo superavano il milione

Un po' di storia 2/2

- I 2 milioni furono toccati in aprile del 2000, e alla fine dello stesso anno erano addirittura 4.800.000. Il 2000 è stato sicuramente l'anno di maggiore crescita del PHP, coincisa anche con il rilascio della versione 4.0.0, con un nuovo motore (Zend) molto più veloce del precedente ed una lunga serie di nuove funzioni, fra cui quelle importantissime per la gestione delle sessioni
- La crescita di PHP, nonostante sia rimasta bloccata fra luglio e ottobre del 2001, è poi proseguita toccando quota 7.300.000 server alla fine del 2001, per superare i 10 milioni alla fine del 2002, quando è stata rilasciata la versione 4.3.0.
- Oggi PHP è conosciuto come **PHP: Hypertext Preprocessor**, ed è un linguaggio completo di scripting, sofisticato e flessibile, che può girare praticamente su qualsiasi server Web, su qualsiasi sistema operativo (Windows o Unix/Linux, ma anche Mac, AS/400, Novell, OS/2 e altri), e consente di interagire praticamente con qualsiasi tipo di database (MySQL, PostgreSQL, Sql Server, Oracle, SyBase, Access e altri)

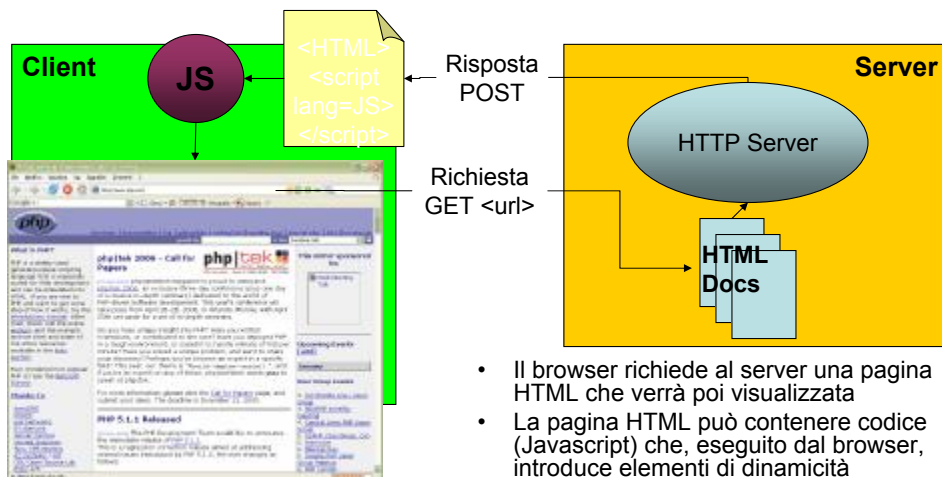
Cos'è

- È un linguaggio di programmazione di tipo scripting (Bash, Tcl, Javascript, Perl, ...):
 - Interpretato (non compilato)
 - Tipizzazione minima (casting implicito)
 - Esecuzione di comandi di shell
- È un linguaggio procedurale con estensioni ad oggetti
- È un linguaggio di programmazione utilizzato per lo sviluppo di pagine web dinamiche "lato server"
- Quando un browser richiede una pagina PHP, il server processa i comandi PHP in essa ed invia i risultati al browser

Dinamicità e Web: Client

- Dinamicità “lato client”
 - Presentazione dinamica:
 - filtraggio dei contenuti ricevuti dal server.
 - maggiore interazione coi contenuti (nuovi media, mappe, menu, interattori, ...).
 - migliore organizzazione dei contenuti e del layout.
 - ...
 - Tecnologie:
 - Javascript.
 - Applet Java.
 - ...

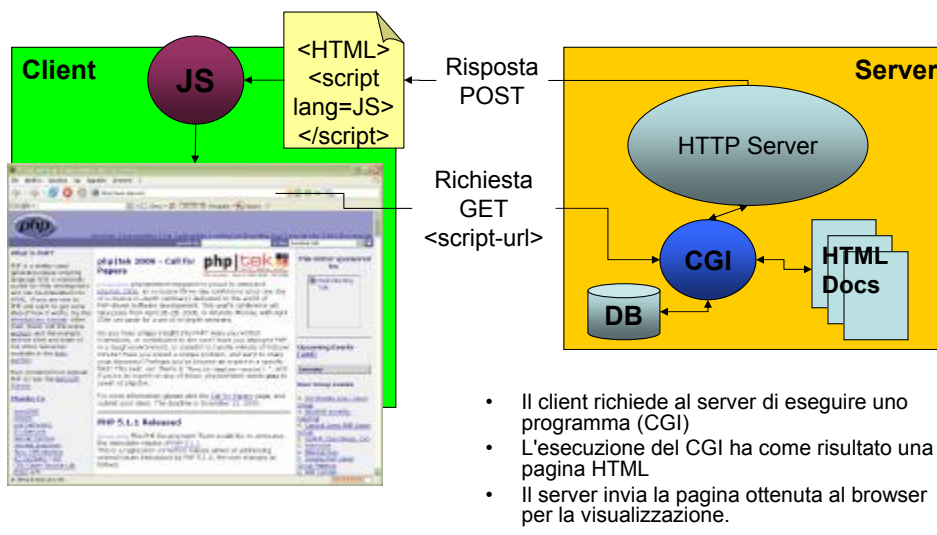
Dinamicità lato Client



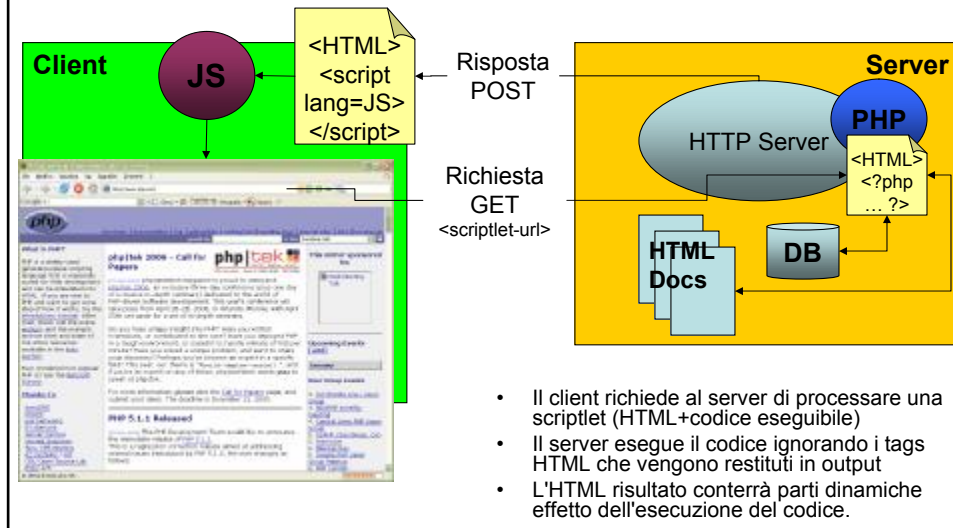
Dinamicità e Web: Server

- Dinamicità “lato server”
 - Contenuti dinamici
 - informazioni prelevate da diverse sorgenti (file, DB, Internet, ...).
 - processamento testo/immagini.
 - gestione profili, sessioni.
 - ...
 - Tecnologie:
 - Common Gateway Interface (CGI).
 - Scriptlet (PHP, Perl, ASP, JSP, ...).

Dinamicità lato Server: CGI



Dinamicità lato Server: scriptlet



CGI e PHP: qualche differenza

Il PHP si sta affermando come linguaggio principe di scripting per la generazione di pagine dinamiche (ASP è proprietario), sostituendo pian piano CGI:

- in primo luogo perchè è multiplatforma, ossia è possibile trovare il motore PHP per le più diffuse piattaforme
- PHP è un linguaggio embedded nel codice HTML delle pagine, e non necessita quindi di ulteriori file esterni per essere eseguito
- uno script PHP, di fatto, non ha bisogno di installazione come avviene per uno script CGI
- con il PHP non si ha più bisogno di particolari configurazioni del webserver in modo da abilitare directory cgi-bin oppure abilitare l'esecuzione di determinati file con determinate estensioni
- le potenzialità del PHP (dalla versione 4) sono praticamente identiche a quelle dei CGI in Perl

Perché scegliere PHP?

Che cosa c'è dietro questo grande successo?

- E' un prodotto Open Source e gratuito, quindi a disposizione di tutti. Il vantaggio garantito dall'Open Source non è limitato al non dover sborsare denaro ma essi generano delle vaste comunità di utenti
- La portabilità è un altro fattore determinante
- Le prestazioni PHP non rimane certo indietro: il motore Zend2, introdotto con PHP 4, ha fornito un formidabile aumento della velocità di esecuzione, portando PHP sugli stessi livelli di ASP, ed in alcuni casi anche oltre
- PHP è facile da imparare: la sua sintassi deriva dal C
- Nonostante la sua semplicità, PHP è però un linguaggio dalle potenzialità vastissime: le sue funzioni infatti partono dalle più tradizionali operazioni di programmazione e ci permettono di realizzare le più svariate funzioni legate al mondo del web: elaborare i moduli html, gestire i cookies, manipolare documenti pdf, etc.
- La possibilità di connettersi ad una vasta gamma di database

PHP e HTML

PHP è un linguaggio di programmazione, immerso nel codice HTML di pagine web, la cui funzione fondamentale è quella di produrre codice HTML condizionato ai risultati di un'elaborazione di informazioni

Realizziamo così il Web dinamico

Tags

- Per inserire codice PHP nelle pagine web si usano speciali tags:

```
<?php ... ?> //il più usato
```

```
<? ... ?> //tag brevi
```

```
<script language="php"> //classico tag Script  
...  
</script>
```

```
<% ... % > //stile ASP
```

N.B. a parte il primo tag gli altri devono essere abilitati nel file di configurazione di PHP: php.ini

Tags PHP e HTML

- Questi tag delimitano il codice PHP, e il codice contenuto al loro interno non sarà inviato al browser, ma eseguito e interpretato
- Da questo potremmo dedurre che tutto ciò che sta **fuori** da questi tag non verrà toccato da PHP, che si limiterà a passarlo al browser così com'è

Un primo esempio

- Per generare il codice da inviare al browser abbiamo due funzioni, che possiamo considerare equivalenti, che sono: `print()` e `echo()` (con o senza parentesi, ogni riga PHP termina con punto e virgola)

```
<HTML>
  <HEAD>
    <TITLE>
  </HEAD>
  <BODY>
    <?php
      print "Buongiorno a tutti!<br>\n";
      echo "E' una bellissima giornata";
    ?>
  </BODY>
```

CODICE HTML

```
<HTML>
  <HEAD>
    <TITLE>Pagina di prova in PHP</TITLE>
  </HEAD>
  <BODY>
```

Formattare il codice e la pagina

Facciamo caso ad un dettaglio: nelle istruzioni in cui stampavamo "*Buongiorno a tutti*", abbiamo inserito, dopo il "`
`", il simbolo "`\n`". Questo simbolo ha una funzione abbastanza importante, anche se non fondamentale, che serve più che altro per dare leggibilità al codice HTML che stiamo producendo

```
<?php
  print("prima riga\n");
  print("seconda riga<br>");
  print("terza riga");
?>
```

CODICE HTML

```
prima riga
seconda riga<br>terza riga
```

OUTPUT

```
prima riga seconda riga
terza riga
```

I Commenti

- Un commento serve per documentare e rendere più chiari gli script:
 - descrivere il contenuto delle variabili utilizzate
 - chiarire il significato dei parametri da passare ad una funzione
 - spiegare cosa si sta facendo in un certo punto del codice (specialmente in caso di script molto lunghi o frammentati)

```
// Commento in stile C++
```

```
# Commento in stile Perl
```

```
/* Questo è un commento in stile C,  
che può occupare più righe ma deve  
essere chiuso con l'apposito simbolo */
```

Le variabili

- Le variabili sono alcuni dei componenti fondamentali di qualsiasi linguaggio di programmazione.
- Il nome di una variabile inizia sempre con dollaro (\$)
- In PHP possiamo scegliere il nome delle variabili usando lettere, numeri e l'*underscore* (_). Il primo carattere del nome deve essere però una lettera o un underscore (non un numero) ed è case sensitive
- PHP non richiede che le variabili vengano dichiarate prima del loro uso

```
$a = 5;
```

- Qualche operazione più complicata con le variabili

```
$a = 9;  
$b = 4;  
$c = $a * $b;
```

Variabili dinamiche

- In qualche situazione può presentarsi la necessità di utilizzare delle variabili senza sapere a priori come si chiamano. In questi casi, il nome di queste variabili sarà contenuto in ulteriori variabili

```
$pippo = 'gawrsh!';
$pluto = 'bau!';
$paperino = 'quack!';

$nome = 'pippo';
print ($$nome.'<br>');
$nome = 'pluto';
print ($$nome.'<br>');
$nome = 'paperino';
print ($$nome.'<br>');
```

CODICE HTML

```
gawrsh!<br>bau!<br>quack!<br>
```

OUTPUT

```
gawrsh!
bau!
quack!
```

Nota: il punto serve a concatenare i valori che vengono stampati

Variabili pre-assegnate

Variabili del Server	Descrizione
<code>\$ SERVER[" PHP_SELF "</code>	Il percorso dello script (da document root)
<code>\$ SERVER[" QUERY_STRING "</code>	Query string della URI richiesta
<code>\$ SERVER[" REQUEST_URI "</code>	URI richiesta del client
<code>\$ SERVER[" SERVER_NAME "</code>	Nome del server
<code>\$ SERVER[" SERVER_PORT "</code>	Port number del server
Variabili di Ambiente	Descrizione
<code>\$ ENV[" PWD "</code>	Directory corrente
<code>\$ ENV[" PATH "</code>	Percorsi per la ricerca di eseguibili e librerie
<code>\$ ENV[" USERNAME "</code>	Nome dell'utente
<code>\$ ENV[" HOME "</code>	Home directory dell'utente
Variabili di GET/POST	Descrizione
<code>\$ HTTP_POST_FILES[file] [...]</code>	Matrice di attributi del <code>file</code> di upload
Variabili Cookies	Descrizione
<code>\$ COOKIE [...]</code>	Array associativo delle coppie <code>cookie - valore</code>
Variabili di Sessione	Descrizione
<code>\$ SESSION[var]</code>	Array associativo delle coppie <code>var = valore</code>

Eliminare una variabile

- In alcune situazioni ci può capitare di avere la necessità di eliminare una variabile: in questo caso PHP ci mette a disposizione l'istruzione `unset()`:

```
unset($variabile);
```

Dopo l'istruzione `unset`, sarà come se la variabile non fosse mai esistita.

I tipi di variabile

- I tipi in PHP si dividono in:
 - Tipi semplici:
 - Booleani
 - Interi
 - Numeri in virgola mobile
 - Stringhe
 - Tipi composti:
 - Array
 - Oggetti

Tipi semplici: booleani

- Le variabili booleane sono le più semplici: il loro valore può essere **TRUE** o **FALSE** (vero o falso)

```
$vero = TRUE;  
$falso = FALSE;
```

Tipi semplici: interi

- Un numero intero, positivo o negativo, il cui valore massimo (assoluto) può variare in base al sistema operativo su cui gira PHP, ma che generalmente si può considerare di circa 2 miliardi (2 elevato alla 31esima potenza).
- Hanno sempre il segno
- Possono essere espressi sia in base 10, in ottale (0...) o in esadecimale (0x...)

```
$int1 = 129;  
$int2 = -715;  
$int3 = 5 * 8;           // $int3 vale 40  
$int4 = $int1 + $int2   // $int4 vale -586
```

Tipi semplici: numeri in virgola mobile

- I numeri in virgola mobile corrispondono in PHP al tipo di dato double
- Le notazioni supportate sono:
 - Notazione in virgola mobile
 - Notazione scientifica con esponente

```
$vm1 = 4.153; // 4,153  
$vm2 = 3.2e5; // 3,2 * 10^5, cioè 320.000  
$vm3 = 4E-8; // 4 * 10^-8, cioè 4/100.000.000 = 0,00000004
```

Tipi semplici: stringhe

- Una stringa è un qualsiasi insieme di caratteri, senza limitazione. Le stringhe possono essere espresse in tre modi:
 - Delimitate da apici (singoli)
 - Delimitate da virgolette (doppie)
 - Con la sintassi heredoc

Stringhe delimitate da apici

- Le stringhe delimitate da apici singoli sono la forma più semplice, consigliata quando all'interno della stringa non vi sono variabili di cui vogliamo ricavare il valore

```
$frase = 'Anna disse: "Ciao a tutti!" ma nessuno rispose';  
print $frase;
```

```
OUTPUT  
Anna disse: "Ciao a tutti!" ma nessuno rispose
```

Stringhe delimitate da virgolette

- Le virgolette ci consentono di usare le stringhe in una maniera più sofisticata, in quanto, se all'interno della stringa delimitata da virgolette PHP riconosce un nome di variabile, lo sostituisce con il valore della variabile stessa (la variabile viene risolta)

```
$nome = 'Anna';  
print "$nome è simpatica... a pochi";  
print '$nome è simpatica... a pochi';
```

```
OUTPUT  
Anna è simpatica... a pochi  
$nome è simpatica... a pochi
```

Stringhe con sintassi heredoc

- Questa sintassi ci consente di delimitare una stringa con i caratteri <<< seguiti da un identificatore (in genere si usa 'EOD', ma è solo una convenzione). Tutto ciò che segue questo delimitatore viene considerato parte della stringa, fino a quando non viene ripetuto l'identificatore seguito da un punto e virgola.

```
$nome = "Paolo";  
$stringa = <<<EOD  
Il mio nome è $nome  
EOD;  
print $stringa;
```

OUTPUT

```
Il mio nome è Paolo
```

```
$frase = "ciao a tutti";  
$stringa = <<<EOT  
Il mio saluto è "$frase"  
EOT;  
print $stringa;
```

OUTPUT

```
Il mio saluto è "ciao a tutti"
```

Attenzione: l'identificatore di chiusura deve occupare una riga a sé stante, deve iniziare a colonna 1 e non deve contenere nessun altro carattere (nemmeno spazi vuoti) dopo il punto e virgola

Tipi composti: array (creazione) 1/2

- Un array è una variabile complessa che contiene una serie di valori ciascuno dei quali caratterizzato da una chiave (indice numerico o alfanumerico)
- E' possibile creare gli array in due modi, tramite la funzione `array()`

```
$colori = array('bianco', 'nero', 'giallo', 'verde', 'rosso');
```

- Oppure lo si può creare implicitamente, aggiungendovi elementi

```
$colori[] = 'bianco';  
$colori[] = 'nero';  
$colori[] = 'giallo';  
$colori[] = 'verde';  
$colori[] = 'rosso';
```

```
$colori[0] = 'bianco';  
$colori[1] = 'nero';  
$colori[2] = 'giallo';  
$colori[3] = 'verde';  
$colori[4] = 'rosso';
```

Tipi composti: array (creazione) 2/2

Nel caso di un array con chiavi associative:

- Mediante la funzione `array()`

```
$cibo = array('es' => "paella",  
            'it' => "pasta",  
            'jp' => "sushimi");
```

- Oppure lo si può creare implicitamente, aggiungendovi elementi

```
$cibo['es'] = "paella";  
$cibo['it'] = "pasta";  
$cibo['jp'] = "sushimi";
```

Tipi composti: array (attenzione)

- **Attenzione:** capita qualche volta di leggere che PHP gestisce due tipi di array: numerici ed associativi. Per array associativi si intendono quelli che hanno chiavi (o indici) in formato stringa, come nell'ultimo esempio che abbiamo visto. In realtà però questo **non è esatto**: infatti PHP gestisce **un unico** tipo di array, le cui chiavi possono essere numeriche o associative. La differenza è sottile, ma significativa: infatti le chiavi numeriche ed associative possono coesistere **nello stesso array**.

```
$formazione[1] = 'Buffon';           $formazione[2] = 'Panucci';  
$formazione[3] = 'Nesta';           $formazione[4] = 'Cannavaro';  
$formazione[5] = 'Coco';            $formazione[6] = 'Ambrosini';  
$formazione[7] = 'Tacchinardi';     $formazione[8] = 'Perrotta';  
$formazione[9] = 'Totti';           $formazione[10] = 'Inzaghi';  
$formazione[11] = 'Vieri';  
$formazione['ct'] = 'Trapattoni';
```

Tipi composti: array (accesso)

- A questo punto ciascuno dei nostri cinque colori è caratterizzato da un indice numerico, che PHP assegna automaticamente a partire da 0

```
print $colori[1]; //stampa 'nero'  
print $colori[4]; //stampa 'rosso'
```

- Mentre per l'array associativo ciascuno dei piatti è caratterizzato da un indice alfanumerico

```
print $cibo['es']; //stampa 'paella'  
print $cibo['jp']; //stampa 'sushimi'
```

- Mentre per l'array misto

```
print $formazione[1]; //stampa 'Buffon'  
print $formazione['ct']; //stampa 'Trapattoni'
```

Tipi composti: array multidimensionali

- Un array a più dimensioni è un array nel quale uno o più elementi sono degli array a loro volta (non per forza tutti con la stessa struttura).

```
$persone = array(  
$persone = array(  
  1 => array('nome' => 'Mario Rossi', 'residenza' => 'Roma',  
  'ruolo' => 'impiegato'),  
  2 => array('nome' => 'Paolo Bianchi', 'data_nascita' =>  
  '1968/04/05', 'residenza' => 'Torino'),  
  'totale_elementi' => 2);  
  
print $persone[1]['residenza']; // stampa 'Roma'  
print $persone['totale_elementi']; // stampa '2'  
print $persone[2]['nome']; // stampa 'Luca'
```

Contare gli elementi di un array

- Se vogliamo sapere di quanti elementi è composto un array, possiamo utilizzare la funzione `count()`

```
$numero_elementi = count($formazione);
```

- Nell'esempio visto prima, la variabile `$numero_elementi` assumerà il valore 12: infatti abbiamo 11 elementi con chiavi numeriche e 1 ('ct') con la chiave associativa.

Eliminazione su array

- Le istruzioni riportate di seguito sono utilizzate per eliminare soltanto un suo elemento o un intero array

```
unset($colori[2]); // elimina l'elemento 'giallo'  
unset($formazione); // elimina l'intero array $formazione
```

- Con la prima di queste due istruzioni abbiamo eliminato l'elemento con chiave 2 dall'array `$colori`. Questo creerà un 'buco' nell'array, che passerà così dall'elemento 1 all'elemento 3. Con la seconda istruzione invece eliminiamo l'intero array `$formazione`.

Gli operatori aritmetici

- Gli operatori sono un altro degli elementi di base di qualsiasi linguaggio di programmazione, in quanto ci consentono non solo di effettuare le tradizionali operazioni aritmetiche, ma più in generale di manipolare il contenuto delle nostre variabili.
- Il più classico e conosciuto degli operatori è quello di assegnazione:

```
$nome = 'Giorgio';
```

```
$a = 5;
```

```
$b = $a;
```

- Altri operatori molto facili da comprendere sono quelli aritmetici: addizione, sottrazione, divisione, moltiplicazione, modulo.

```
$a = 3 + 7; //addizione           $b = 5 - 2; //sottrazione  
$c = 9 * 6; //moltiplicazione     $d = 8 / 2; //divisione  
$e = 7 % 4; //modulo
```

Gli operatori di incremento e decremento

- Operatori ancora più sintetici che si possono utilizzare per incrementare o decrementare i valori di una unità

```
//incrementa di 1: equivale ad $a = $a + 1, o $a += 1  
$a++;  
++$a;  
  
//decrementa di 1: equivale ad $a = $a - 1, o $a -= 1  
$a--;  
--$a;
```

Gli operatori per le stringhe

- Uno degli operatori più utilizzati è quello che serve per concatenare le stringhe: il punto.

```
$nome = 'pippo';  
$stringa1 = 'ciao '.$nome; //$stringa1 vale 'ciao pippo'
```

```
$a = 'pippo';  
$x = 'ciao';  
$x .= $a; //La stringa $a viene concatenato a $x
```

Gli operatori di confronto

- Gli operatori di confronto sono fondamentali perché ci permettono, effettuando dei confronti fra valori, di prendere delle decisioni, cioè di far svolgere al nostro script determinate operazioni invece di altre

ESEMPI NUMERICI

ESEMPI SU STRINGHE

```
$a = 'Mario'; $b = 'Giorgio'; $c = 'Giovanni'; $d = 'antonio';  
$e = '4 gatti';
```

```
$a < $b; //falso, la 'G' precede la 'M'
```

```
$b < $c; //vero, la 'r' ('Gior') precede la 'v' ('Giov')
```

```
$d > $a; //vero, la 'a' minuscola è 'maggiore' delle maiuscole
```

```
$c > $e; //vero, ogni lettera è 'maggiore' di qualsiasi cifra
```

```
$c <= $b; //vero
```

- >= : maggiore o uguale
- < : minore
- <= : minore o uguale

Gli operatori logici

- Con gli operatori logici possiamo combinare più valori booleani, oppure negarne uno (nel caso di NOT). Questi valori sono:
 - **or**: valuta se almeno uno dei due operatori è vero; si può indicare con 'Or' oppure '||'
 - **and**: valuta se entrambi gli operatori sono veri; si indica con 'And' o '&&'
 - **xor**: viene chiamato anche 'or esclusivo', e valuta se **uno solo** dei due operatori è vero: l'altro deve essere falso; si indica con 'Xor'
 - **not**: vale come negazione e si usa con un solo operatore: in pratica è vero quando l'operatore è falso, e viceversa; si indica con '!'

ESEMPI

```
10 > 8 And 7 < 6; //falso, la prima è vera - la seconda è falsa
10 > 8 Or 7 < 6; //vero
9 > 5 And 5 == 5; //vero: entrambe le condizioni sono vere
9 > 5 Xor 5 == 5; //falso: solo una delle due deve essere vera
4 < 3 || 7 > 9; //falso: nessuna delle due condizioni è vera
6 == 6 && 1 > 4; //falso: solo la prima condizione è vera
```

Espressioni

- PHP si può definire un linguaggio orientato alle espressioni
- Un'espressione è "qualsiasi cosa che abbia un valore"
- Il procedimento che, a partire da una data espressione, restituisce il valore corrispondente viene definito *valutazione dell'espressione*
- Sono espressioni:
 - tipi scalari (costanti booleane, numeri interi ed in virgola mobile, stringhe)
 - tipi composti (array ed oggetti)
 - le variabili
 - le funzioni, con valore quello restituito dalla funzione stessa
 - combinazione di espressioni con operatori (booleani, di confronto, ...)
- Altre espressioni: l'assegnazione ($\$n=2$) e l'espressione condizionale ($\text{expr1} ? \text{expr2} : \text{expr3}$)

Strutture di controllo: condizioni (if-then-else)

- Con le strutture di controllo andiamo ad analizzare un altro degli aspetti fondamentali della programmazione: la possibilità cioè di eseguire operazioni diverse, ed eventualmente di eseguirle più volte
- La principale di queste istruzioni è la *if-then-else*

```
if ($nome == 'Luca') {  
    print "bentornato Luca!";  
}  
elseif ($cognome == 'Verdi') {  
    print "Buongiorno, signor Verdi";  
}  
else {  
    print "ciao $nome!";  
}
```

Strutture di controllo: condizioni (switch-case)

- Passiamo ora a verificare una seconda istruzione che ci permette di prevedere diversi valori possibili per un'espressione:

```
switch ($nome) {  
    case 'Luca':  
        print "E' tornato Luca!";  
        break;  
    case 'Mario':  
        print "Ciao, Mario!";  
        break;  
    case 'Paolo':  
        print "Finalmente, Paolo!";  
        break;  
    default:  
        print "Benvenuto, chiunque tu sia";  
}
```

Strutture di controllo: cicli (for)

- I cicli sono un altro degli elementi fondamentali di qualsiasi linguaggio di programmazione, in quanto ci permettono di eseguire determinate operazioni in maniera ripetitiva.
- Il costrutto *for* ci consente di eseguire il ciclo un numero di volte prefissato, utilizzando un indice

```
for ($mul = 1; $mul <= 10; $mul++)  
{  
    $ris = 5 * $mul;  
    print("5 * $mul = $ris<br>");  
}
```

Strutture di controllo: cicli (while, do-while)

- Il costrutto *while* si può considerare come una specie di *if* ripetuto più volte: infatti la sua sintassi prevede che alla parola chiave *while* segua, fra parentesi, la condizione da valutare, e fra parentesi graffe, il codice da rieseguire

```
$mul = 1;  
while ($mul <= 10) {  
    $ris = 5 * $mul;  
    print("5*$mul=$ris<br>");  
    $mul++;  
}
```

```
$mul = 11;  
do {  
    $ris = 5 * $mul;  
    print("5*$mul=$ris<br>");  
    $mul++;  
} while ($mul <= 10)
```

Strutture di controllo: cicli (foreach)

- Quest'ultimo particolare tipo di ciclo, é pensato appositamente per il trattamento degli array. Questo ci permette di costruire un ciclo che viene ripetuto per ogni elemento dell'array che gli passiamo.

```
foreach ($arr as $chiave => $valore) {  
    ...istruzioni da ripetere....  
}
```

```
foreach ($arr as $valore) {  
    ...istruzioni da ripetere....  
}
```

Strutture di controllo: cicli Uscire da un ciclo

- Abbiamo a disposizione due strumenti per modificare il comportamento del nostro script dall'interno del ciclo:
 - **continue** (non completa la presente iterazione e passare alla successiva)
 - **break** (interrompe definitivamente l'esecuzione del ciclo)

```
for ($ind = 1; $ind < 500; $ind++) {  
    if ($ind % 100 == 0) {  
        break;  
    }elseif ($ind % 25 == 0) {  
        continue;  
    }  
    print("valore: $ind<br>");  
}
```

Funzioni

- La dichiarazione di funzioni avviene con la parola chiave **function** e consente di estendere il linguaggio, aggiungendovi nuovi comandi

```
function nome-funzione (argomento, ...) {  
    blocco-istruzioni;  
}
```

- *nome-funzione* è il nome da assegnare alla funzione e con cui essa verrà invocata
- *argomento,...* è un elenco, eventualmente vuoto, di argomenti (o parametri) da fornire alla funzione
- *blocco-istruzioni* rappresenta il corpo della funzione (qualunque codice PHP valido, persino altre funzioni e definizioni di classe)
- Il risultato viene restituito con *return(valore)* che provoca l'uscita dalla funzione (può essere restituito qualsiasi tipo, incluse liste e oggetti)
- L'esecuzione dello script riprende dal punto in cui essa era stata invocata

Funzioni: cosa accade?

- Quando viene chiamata una funzione:
 - PHP cerca la funzione per nome (se non la trova genera un errore)
 - PHP sostituisce le variabili nell'elenco di parametri della definizione con i valori degli argomenti (passaggio per valore o per riferimento)
 - Vengono eseguite le istruzioni nel corpo della funzione, restituendo l'eventuale valore se viene eseguita l'istruzione return

```
function maggiorenne($a) {  
    if ($a >= 18) $res = TRUE;  
    else $res = FALSE;  
    return($res);  
}  
  
//La variabile $eta è già definita  
if (maggiorenne($eta)) echo("Puoi accedere al sito<BR>");  
else echo("Non puoi accedere<BR>");
```

Argomenti di Funzioni

- PHP è tollerante rispetto al numero di parametri passati alla funzione (di conseguenza non possono esistere due funzioni con lo stesso nome)
- Una funzione può definire valori (solo costanti) pre-definiti per gli argomenti (gli argomenti pre-definiti devono sempre seguire tutti quelli non pre-definiti)

```
function anagrafe($nome, $cf='non disponibile') {  
    print "Nome: $nome<br>";  
    print "Codice fiscale: $cf<br>";  
}
```

- Di default, gli argomenti della funzione sono passati per valore (se cambiate il valore dell'argomento nel corpo della funzione, esso non cambierà al di fuori)
- Per passare ad una funzione un argomento per riferimento, dovete anteporre un *ampersand* (&) al nome dell'argomento nella definizione della funzione:

```
function aggiungi_qualcosa(&$string) {  
    return $string .= 'e qualche altra cosa.';  
}  
  
$str = 'Questa stringa, '  
echo aggiungi_qualcosa($str);  
// l'output sarà 'Questa stringa, e qualche altra cosa.'
```

Funzioni variabili

- PHP supporta il concetto di funzioni variabili
- PHP cercherà una funzione con lo stesso nome del valore della variabile e cercherà di eseguirla

```
function foo1() {  
    echo "Eseguo foo1()<br>\n";  
}  
function foo2($arg1 = "null", $arg2 = "null") {  
    echo "Eseguo foo2($arg1,$arg2)<br>\n";  
}  
$array_foo = array('a'=> 'foo1', 'd'=> 'foo2');  
foreach ($array_foo as $funzione) {  
    $funzione("hello", "world");  
}
```

Funzioni: variabili e visibilità

- Le variabili all'interno di una funzione sono sempre private, tranne quando sono definite con la parola chiave *global*
- Le funzioni non tengono memoria della loro esecuzione a meno dell'uso della parola chiave *static*
- Una funzione definita in uno script è disponibile ovunque in esso
- Solitamente le funzioni sono definite in file separati (librerie) da includere all'occorrenza nel programma
 - `include(filename)` (e `require(filename)`) include un file e lo valuta
 - I costrutti sono diversi solo per il fatto che `include` ritorna un warning in caso di fallimento, `require` ritorna un fatal error
- Se l'inclusione si verifica dentro una funzione tutto il codice nel file chiamato seguirà lo scope della funzione

OOP: definizioni generali

- I concetti base della OOP: *classe* ed *oggetto* (o istanza): una classe è l'astrazione di un oggetto, mentre un oggetto è istanza di una classe
- La classe definisce un nuovo tipo di dato, caratterizzato da:
 - Un insieme di variabili, detti attributi (proprietà)
 - Un insieme di funzioni che operano su di essi, detti metodi
- Caratteristiche della OOP:
 - Incapsulamento dati – i dati interni di una classe sono trasparenti al programmatore, che opererà su di essi solo tramite i metodi
 - Protezione dati e metodi – l'accesso a dati e metodi "privati" di una classe è disciplinato secondo diverse politiche (vari livelli di protezione)
 - Interfaccia – i metodi rappresentano, nella semantica e non nella implementazione, l'interfaccia per usare la classe
 - Modularizzazione – il programma viene decomposto in moduli, che possono essere stati sviluppati da altri utenti (Riuso del codice)
 - Ereditarietà – le classi ereditano metodi e attributi da altre classi (Riuso del codice)

OOP: proprietà e metodi

ESEMPIO CLASSE CARRELLO

```
class Carrello {
    var $items; // Articoli nel carrello
    // Aggiunge $num articoli di $artnr nel carrello
    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
    }
    //Rimuove $num articoli di $artnr dal carrello
    function remove_item ($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return TRUE;
        } else {
            return FALSE;
        }
    }
}
```

OOP: costruttore

- La parola chiave **new** ci consente di istanziare una classe, cioè creare un oggetto istanza di una classe
- Il costruttore di classe è il metodo, con lo stesso nome della classe, che è richiamato automaticamente alla creazione di un'istanza della classe
- Se all'interno di una classe base non è presente un costruttore, in automatico viene invocato quello della classe padre
- Esiste in PHP 5 un nome standardizzato per i costruttori, `__construct()`, così se la classe dovesse essere rinominata non sarà necessario modificare anche il nome del suo costruttore

OOP: distruttore

- In PHP5 è stato introdotto il distruttore `__destruct()`: la funzione che viene richiamata automaticamente quando la classe viene distrutta
- Non è possibile richiamare un distruttore di una classe padre
- I distruttori sono utili per "operazioni di pulizia", come ad esempio eliminare file temporanei e chiudere le connessioni a database

OOP: ereditarietà

- È possibile generare classi per estensione di altre classi
- Una classe estesa o derivata ha tutte le variabili e le funzioni della classe di base (ereditarietà) più tutto ciò che viene aggiunto dall'estensione
- Una classe estesa dipende sempre da una singola classe di base (no ereditarietà multipla)
- Le classi si estendono usando la parola chiave **extends**

```
class Carr_con_nome extends Carrello
```

Eccezioni: throw-try-catch

- Con l'avvento del PHP5 è possibile gestire le eccezioni ovvero è possibile gestire le situazioni di errore in modo più evoluto
- Le eccezioni vengono definite dalla terna:
 - **throw**: genera l'eccezione (nuova istanza nella classe eccezione o una estesa)
 - **try**: esegue un blocco di istruzioni con abilitato il meccanismo di cattura delle eccezioni
 - **catch**: specifica il blocco di istruzioni da eseguire in caso di eccezione (gestore dell'eccezione)

Eccezioni: esempio

```
function a() {
    echo "dentro a()\n";
    echo " Emetto l'exception \n";
    throw new Exception ("hei");
    echo "esco da a();\n";
}

try {
    a();
    echo "Dopo a()\n";
} catch (Exception $e) {
    echo " Eccezione:". $e->getMessage();
    echo "\n";
}

echo "finito";
```

OUTPUT

```
dentro a()
Emetto l'exception
Eccezione:hei
finito
```

PHP5: Riferimenti

- www.php.net
- www.apache.org
- freephp.html.it
- pear.php.net
- pecl.php.net