Definizione di PHP

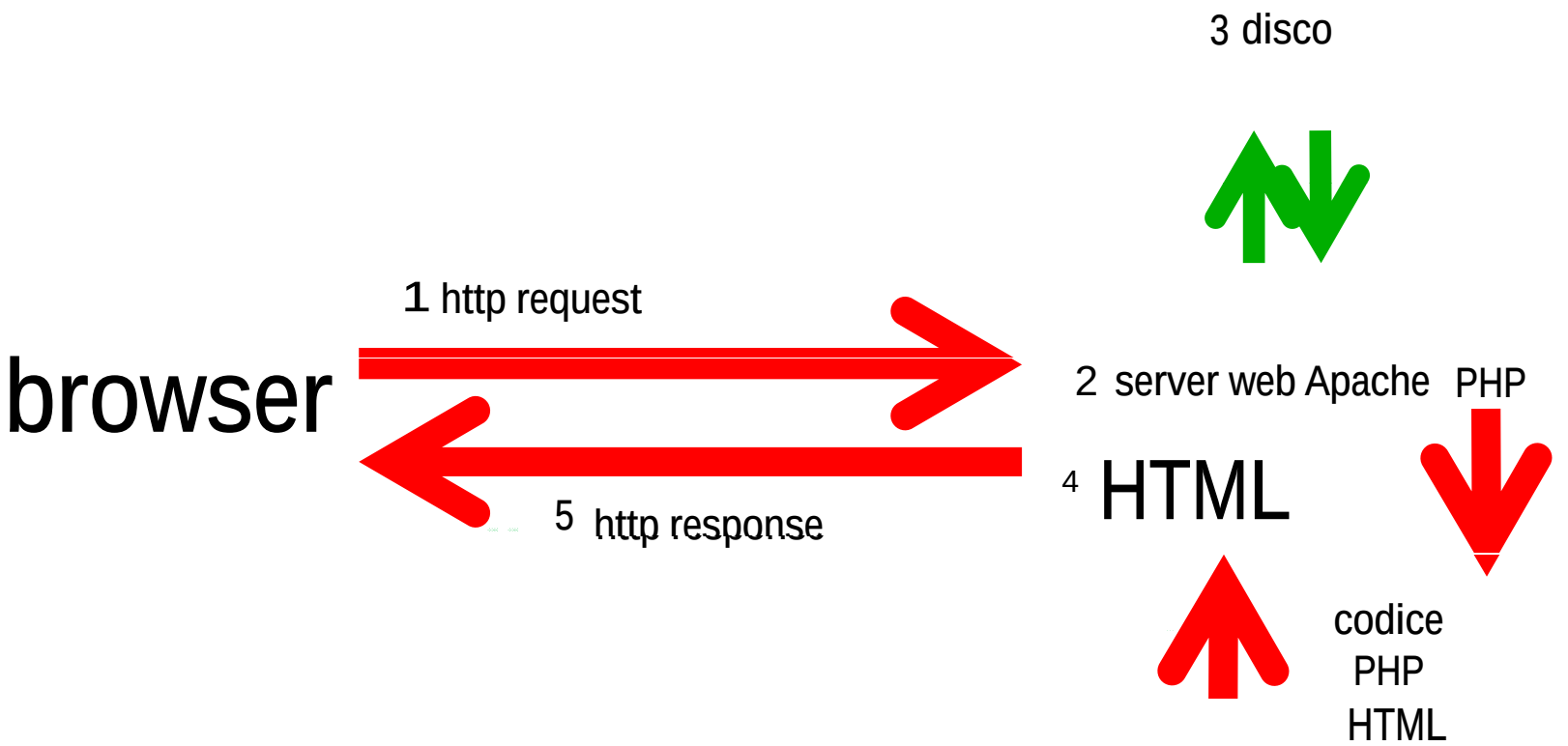
- *PHP è un linguaggio di **scripting interpretato**, con **licenza open source**, originariamente concepito per la realizzazione di **pagine web dinamiche.***
- *Attualmente è utilizzato principalmente per sviluppare **applicazioni web lato server** ma può essere usato anche per scrivere script a linea di comando o applicazioni stand-alone con interfaccia grafica.*
- *Il suo nome è un acronimo ricorsivo che sta per PHP: Hypertext Preprocessor (PHP: preprocessore di ipertesti).*
- *Il software MediaWiki, che gestisce i siti del progetto Wikipedia, è scritto in PHP.*

*Definizione tratta da **it.wikipedia.org***

Caratteristiche di PHP

- Linguaggio di scripting interpretato
  Il codice sorgente non viene compilato ma interpretato
- Rilasciato con licenza Open Source
  E' rilasciato con licenza open source (PHP License).
- Capace di creare pagine web dinamiche
  Permette la produzione di pagine che cambiano a seconda degli input o delle condizioni al contorno
- Linguaggio per applicazioni web lato server
  Il codice è eseguito sul server e non nel browser
- Linguaggio multipiattaforma
  E' disponibile per diversi sistemi operativi tra cui Windows, Linux, Mac OS X, Solaris, BSD, AS/400, OS/2. La disponibilità del codice agevola il porting.

Come viene processato il codice



Lavorare con PHP

Per lavorare con PHP occorre:

- Web server (tipicamente Apache o IIS)
- Interprete PHP
- DBMS (opzionale, ma spesso indispensabile)

Esistono pacchetti software integrati

Integrano l'interprete, il server web e il database.

Vantaggi: facilità di installazione e d'uso.

Svantaggi: eccessiva rigidità

Sito ufficiale di PHP: http://www.php.net/

Installazione

Sequenza di installazione:

1. Apache
2. PHP

Strumenti per scrivere il codice

- Editor di file testo evoluti
  - Highlight della sintassi
  - Supporto alla scrittura del codice
  - Segnalazione di errori
  - *Notepad2, Notepad++, Textpad, UltraEdit, jEdit*

PHP e HTML

- Il codice è embedded nell'HTML

  Il codice PHP si integra nel codice HTML tramite appositi tag che lo racchiudono.

- **Le istruzioni PHP terminano sempre con ;**

```
<BODY>
  <P>Questa è una pagina</P>
  <?php
       $messaggio = ”Esempio di codice”;
       echo $messaggio;
  ?>
</BODY>
```

Tag di apertura e chiusura

E' possibile specificare quattro diversi tipi di tag per racchiudere il codice PHP

```
<?php echo "forma più corretta per racchiudere
istruzioni PHP \n"; ?>

<? echo "forma più semplice per racchiudere istruzioni
PHP \n"; ?>
<?= echo "forma equivalente" ?>

<script language="php"> echo "altra forma permessa da
HTML, alla maniera di Javascript"; </script>;

<% echo "Alla maniera dei tag in stile ASP"; %>
<%= echo "Forma equivalente"; %>
```

Variabili

*Una **variabile** identifica una **porzione di memoria** destinata a **contenere dei dati**, che possono essere **modificati** nel corso dell'esecuzione di un programma. Una variabile è spesso, ma non sempre, caratterizzata da un **nome** (inteso solitamente come una sequenza di caratteri e cifre).*

*Definizione tratta da **it.wikipedia.org***

Variabili in PHP

- Non è necessario dichiararle “a priori”
- Non è necessario indicarne esplicitamente il tipo
- Vengono allocate dinamicamente quando utilizzate
- Identificate da un nome che inizia con il carattere $

Variabili - esempio

```
<?php
// a $num viene assegnato
// un valore numerico
$num = 10;
// a $stringa viene assegnato un
// valore alfanumerico
$stringa = ”questa è una stringa”;
// a $str2 viene assegnata
// la stringa “1”
$str2 = ”1”;
// a $num2 viene assegnato un
// valore numerico decimale
$num2 = 5.11
?>
```

Tipi di dati

Tipi di dati scalari in PHP

- **logico (booleano):** vero o falso {TRUE, FALSE}
- **intero:** $z \in \mathbb{Z} \mid \mathbb{Z} = \{...,-2,-1,0,1,2,...\}$
  Limiti: generalmente 32 bit cioè $(-2^{31}) \leq z \leq (2^{31}-1)$
- **decimale:** $q \in \mathbb{Q} \mid q=x/y\ x,y \in \mathbb{Z}$ e $y \neq 0$
  Limiti: generalmente ~1.8e308, ~14 cifre decimali

  A causa dei limiti fisici del calcolatore non è possibile rappresentare numeri irrazionali ($\sqrt{2}$, $\sqrt{3}$) o numeri trascendenti (π, e) che vengono approssimati

  Es: 3.14159265 Il separatore decimale è il punto (.)
- **stringa:** sequenza di caratteri alfanumerici delimitata da apici o virgolette

Tipi di dati - esempio

```
<?php
// valori logici
$vero = true;
$falso = false;

// valori numerici interi
$num1 = 0;
$num2 = 13;
$num3 = -5;

// valori numerici decimali
$str1 = -2.1245478;
$str2 = 3.14159265358793238;
$str3 = 0.11;
?>
```

Tipi di dati - stringhe

Differenza tra stringhe delimitate da virgolette e quelle delimitate da apici

```
<?php
$str = “tutti voi”;
echo “Ciao a $str”; //Stampa “Ciao a tutti voi”
echo 'Ciao a $str'; // Stampa “Ciao a $str”
?>
```

Caratteri Escape

Stringhe delimitate da virgolette (“”)			
\n	linefeed	\\	backslash
\r	carriage return	\$	dollar sign
\t	horizontal tab	\"	double-quote

Stringhe delimitate da apici ('')	
\'	single-quote

Commenti nel codice

Analogamente ad altri linguaggi il PHP permette di scrivere **commenti** all'interno del codice.

```
<?php
// questo è un commento su una riga
echo “messaggio dopo il primo commento”;
/* Questo è un commento
      articolato su più righe */
echo “messaggio dopo il secondo commento”;
echo “un altro”; # Questo è un altro commento
?>
```

I commenti hanno effetto solo nei blocchi contenenti codice PHP, cioè tra i tag **<?php ?>** ed equivalenti.

Costanti simboliche

- Una costante simbolica è un elemento il cui **valore non muta e non può mutare** durante tutta l'esecuzione di uno script
- Le costanti hanno **visibilità globale** all'interno del codice
- Il nome di una costante non può essere preceduto dal simbolo $, perché questo identifica le variabili.
- Definizione di costanti:
  `define ("nome_costante",valore, [case sensitive]);`
- Per l'utilizzo di costanti, ci si riferisce al nome senza virgolette o apici.
  `echo nome_costante;`

Costanti simboliche - 2

Per verificare se una costante è stata definita in precedenza viene utilizzata la funzione *defined()* che restituisce un valore booleano

```
<?php
   // definisce la costante “cost”
   // con valore 30
   define (“cost”,30);

   echo cost; // stampa il valore di cost

   // restituisce TRUE perché la costante
   // è già stata definita
         defined('cost');
?>
```

Operatori

*Un **operatore** è un elemento di aggregazione che applicato ad uno o due **operandi** compone una espressione che genera un **risultato**.*

Operatore **unario**: applicato ad un solo elemento
Operatore **binario**: applicato a due elementi

Un operatore speciale è l'operatore di **assegnazione =**
Esempi: $a = 1; $b = $a; $c = “pippo”;

Tipi di operatori:

- Operatori aritmetici
- Operatori relazionali
- Operatori logici
- Operatori su stringhe

Operatori aritmetici

Gli **operatori aritmetici** permettono l'esecuzione di **operazioni matematiche** su operandi.

Negazione:	-$a
Somma:	$a + $b
Sottrazione:	$a - $b
Moltiplicazione:	$a * $b
Divisione:	$a / $b
Modulo:	$a % $b

- L'operatore di divisione ("/") restituisce un valore decimale anche se i due operandi sono interi.
- L'operatore modulo (“%”) restituisce il resto della divisione intera tra gli operandi. Il risultato di $a % $b è negativo se $a è negativo.

Operatori su stringhe

Gli operatori su stringhe permettono la manipolazione di dati di tipo stringa.

Concatenazione: $a = $b . $c;
assegna a $a la stringa composta dall'unione del contenuto di $b seguito da quello di $c

```
<?php
$str1 = “Pippo, Pluto ”;
$str2 = “e Paperino”;
$str  = $str1 . $str2;
echo “Ecco ” . $str;
// visualizza la stringa:
// “Ecco Pippo, Pluto, e Paperino”;
?>
```

Notazione heredoc per stringhe

Notazione heredoc: stringa delimitata all'inizio da <<< seguita da una etichetta e alla fine dalla medesima etichetta seguita da ; (nell'esempio l'etichetta è DATI). L'etichetta che delimita la fine della stringa DEVE essere posizionata nella prima colonna della riga.

```
<?php
$nome='Mario';
$cognome='Rossi';
echo
<<<DATI
Nome: $nome
<br>
Cognome : $cognome
DATI;
/* L'etichetta deve essere posizionata a colonna 1
   e non deve essere seguita da nessun carattere,
   compreso lo spazio */
?>
```

Operatori di incremento

Gli operatori di incremento e decremento sono operatori unari che incrementano o decrementano una variabile. Non agiscono sui valori di tipo boolean. Decrementare il valore NULL non ha effetti, ma incrementarlo dà come risultato 1.

Pre-incremento: ++$a Prima incrementa $a di una unità, poi restituisce il valore di $a (già incrementato)

Post-incremento: $a++ Restituisce il valore di $a, poi incrementa $a di una unità

Pre-decremento: --$a Prima decrementa $a di una unità, poi restituisce $a (già decrementato)

Post-decremento: $a-- Prima restituisce $a, poi decrementa $a di una unità

Operatori speciali di assegnazione

Oltre all'operatore di base assegnazione esistono gli **operatori di assegnazione combinati**, applicabili ad operatori binari sia aritmetici che di tipo stringa.

$a += 5; // equivale a $a = $a + 5;
$a -= 5; // equivale a $a = $a - 5;
$a *= 5; // equivale a $a = $a * 5;
$a /= 5; // equivale a $a = $a / 5;
$a %= 5; // equivale a $a = $a % 5;
$a .= “ciao!”; // equivale a $a = $a . “ciao!”;

```
<?php
$x = 5;
$x += 2; // $x vale 7
$y = "Pippo, ";
$y .= "Pluto!"; // $y vale "Pippo, Pluto!"
?>
```

Operatori relazionali

Gli **operatori relazionali** permettono il **confronto** fra i valori assunti dagli operandi, restituendo un valore di tipo booleano (TRUE, FALSE). Gli operandi possono essere di qualunque tipo.

Uguale:	$a == $b
Diverso:	$a != $b $a <> $b
Identico:	$a === $b
Non identico:	$a !== $b
Maggiore:	$a > $b
Maggiore uguale:	$a >= $b
Minore:	$a < $b
Minore uguale:	$a <= $b

L'operatore === restituisce TRUE se $a è uguale a $b, ed essi sono dello stesso tipo

Operatori logici

Gli operatori logici permettono l'esecuzione di operazioni booelane, restituendo un valore di tipo booleano (TRUE, FALSE).

- Not: ! $a

restituisce FALSE se l'operatore è TRUE

- And: $a && $b oppure $a and $b

restituisce TRUE se entrambi gli operatori sono TRUE

- Or: $a || $b oppure $a or $b

restituisce TRUE se almeno uno dei due operatori è TRUE

- Xor: $a xor $b

restituisce TRUE se solo uno dei due operatori è TRUE

Valutazione di operatori

Esiste un ordine di precedenza tra gli operatori, che indica all'interprete in quale ordine svolgere ogni singola operazione all'interno di una espressione complessa.
Tale ordine può essere modificato utilizzando il simbolo () parentesi. Le operazioni all'interno delle parentesi hanno la precedenza sulle altre. Ecco l'ordine di precedenza:

```
Incremento e decremento ++  --
Moltiplicazione, divisione, modulo *  /  %
Addizione, sottrazione, concatenazione +  -  .
Relazionali <  <=  =>  >
Uguaglianza, disuguaglianza ==  ===  !=  <>  !==
Operatore logico &&
Operatore logico ||
Assegnazione combinata =  +=  -=  /=  *=  %=  .=
Operatore logico and
Operatore logico xor
Operatore logico or
```

Strutture di controllo - IF

Le **strutture di controllo** permettono di modificare il flusso di esecuzione del codice in funzione del verificarsi (o meno) di determinate condizioni.

IF semplice

```
if (condizione) {
istruzioni
}
```

IF con ELSE

```
if (condizione) {
istruzioni
} else {
istruzioni
}
```

IF con ELSEIF

```
if (condizione) {
istruzioni
} elseif (condizione){
istruzioni
} else {
istruzioni
}
```

IF sintassi ridotta

```
(condiz.) ? val1 : val2
```

IF - esempi

```
<?php
$a = 5;
if ($a == 5) {
        print “il valore di \$a è 5”;
} else {
        print “il valore di \$a è diverso da 5”;
}
?>
```

```
<?php
$a = 6;
if ($a > 5) {
        print “il valore di \$a è maggiore di 5”;
} elseif ($a < 5) {
        print “il valore di \$a è minore di 5”;
} else {
        print “il valore di \$a è uguale a 5”;
}
?>
```

```
<?php
$a = ($b >= 0) ? “positivo o nullo” : “negativo”;
?>
```

IF – operatore ternario

La sintassi ridotta del costrutto **IF** viene spesso riferita come **operatore ternario**. <val1> <val2> possono essere valori o funzioni che comunque restituiscono un valore il quale può essere assegnato ad una variabile che precede il costrutto.

Operatore ternario

(condizione) ? val1 : val2

```
<?php
// esempio corretto: print restituisce il valore 1
($b >= 0) ? print "positivo" : print "negativo";

// esempio errato: echo non produce alcun valore
($b >= 0) ? echo "positivo" : echo "negativo";

echo "Il numero maggiore è: ".(($a > $b)?$a:$b);
?>
```

Strutture di controllo - SWITCH

$variabile viene confrontata con i singoli valori. Se una delle condizioni è verificata viene eseguito il blocco di codice relativo finché non è incontrata l'istruzione break, altrimenti viene eseguito il blocco di codice di default.

```
switch ($variabile) {
case valore 1:
istruzioni
break;
case valore 2:
istruzioni
break;
....
default:
istruzioni;
break;
}
```

SWITCH - esempi

```
<?php
$i = 2;
// esempio con switch
switch($i){
   case 0:
      echo Zero“;
   break;
   case 1:
      echo "Uno“;
   break;
   case 2:
      echo "Due“;
   break;
   default:
      print
"altro"
break;
}
?>
```

```
<?php
$i = 2;
// if equivalente
// al costrutto switch
// a fianco
if ($i == 0) {
    echo "Zero";
} elseif  ($i == 1) {
    echo "Uno";
} elseif  ($i == 2) {
    echo "Due";
} else {
    echo "altro";
}
?>
```

Cicli: for, while, do..while

I cicli vengono comunemente usati per ripetere l'esecuzione di un blocco di codice finché non si verifica una determinata condizione. Si differenziano per le modalità temporali ed operative con le quali viene controllato il verificarsi della condizione.

ciclo for

```
for(inizializzazione ; condizione ; incremento)
<istruzioni>
```

ciclo while

```
while(condizione)
istruzioni
```

ciclo do..while

```
Do
istruzioni
while (condizione)
```

Cicli - esempi

```
<?php
for($i = 0;$i <= 10;$i++)  {
    echo $i;
}
?>
```

```
<?php
$i = 0;
while ($i <= 10)
{
    echo $i;
    $i++;
}
?>
```

```
<?php
$i = 0;
do {
    echo $i;
    $i++;
} while ($i <= 10);
?>
```

Funzioni utente

*Una **funzione** è un blocco di codice **identificato da un nome**, richiamabile in qualsiasi punto del programma, in grado di **svolgere determinate operazioni**. Può accettare dei **parametri** in ingresso e produrre un valore come **risultato**. Una funzione può essere **predefinita nel linguaggio** o **definita dall'utente**.*

```
// definizione della funzione
function nomefunzione(arg1,arg2,...,argN) {
     istruzioni
     return valore;
}

// utilizzo della funzione
variabile = nomefunzione();
```

Funzioni utente - esempio

```
<?php
    function quadrato($val) {
       return $val * $val;
    }

    function pitagora($cat1, $cat2) {
       $ipotenusa = sqrt(quadrato($cat1) + quadrato($cat2));
       return $ipotenusa;
    }

    function fattoriale($val) {
       if ($val <= 1)
          return 1;
       else
          return $val*fattoriale($val - 1);
    }

    echo quadrato(5);   //utilizza la funzione quadrato
    echo pitagora(3,4); //utilizza la funzione pitagora
    echo fattoriale(4); //utilizza la funzione fattoriale
?>
```

Array

*Un **array** è una struttura complessa formata da elementi ciascuno dei quali è individuato dal **nome** dell'array e da un **indice** che ne individua la posizione; il primo elemento ha posizione 0.*

```
<?php
    //Crea un array di 4 elementi
    $vet1 = array(“a”,“b”,“c”,“d”);
    // aggiunge un elemento all'array
    // se non esiste ne crea uno nuovo
    $vet1[] = “e”;
    echo $vet[2]; // stampa “c”
    echo $vet[0]; // stampa “a”
    echo $vet[4]; // stampa “e”
    // l'array ha 5 elementi con indici da 0 a 4
    echo $vet[8]; // non stampa niente e non dà errore
?>
```

Array - esempi

```
<?php
// definizione di array
$vet = array("Primo","Secondo","Terzo","Quarto");
for($i=0;$i<10;$i++) {
    echo $vet[$i];
}

// array multidimensionale a valori misti
// il terzo elemento è l'array $vet
$vet1 = array(0,1,$vet,3,TRUE);
for($i=0;$i<5;$i++) {
    print_r($vet1[$i]);
}
?>
```

Array associativi

*Un **array associativo** è una **array** in cui gli **indici** che individuano i singoli elementi dell'array possono essere anche di tipo **stringa**. Non si parla più di coppie (indice, valore) ma di **coppie (chiave, valore)**.*

```
<?php
    // crea un array di un elemento con
    // chiave “primo” e valore “val uno”
    $vettore[“primo”] = “val uno”;

    // array di tre elementi
    $mesi['gen'] = 31;
    $mesi['feb'] = 28;
    $mesi['mar'] = 31;
    // definizione equivalente
    $mesi = array(‘Gennaio’=>31,’Febbraio’=>28,’Marzo’=>31);

    //un array può avere chiavi e valori di tipo eterogeneo
    $vet = array(1=>”primo elem.”,'due'=>2,'tre'=>'terzo');
?>
```

Ciclo foreach

Il ciclo foreach è stato introdotto per facilitare le operazioni con gli array.

le due forme del ciclo foreach

```
foreach(array_expression as $valore)
    istruzione
foreach(array_expression as $chiave => $valore)
    istruzione
```

Cicli for per la stampa di vettore e foreach equivalente

```
for($i=0; $i<count($vet); $i++) {
       print $vet[$i];
}
```

```
foreach($vet as $elem) {
       print $elem;
}
```

Foreach - esempi

```
<?php
        /* definizione di array associtivo.
              Notare che l'istruzione è distribuita
              su più righe
        */
        $mesi = array('Gennaio'=>31,'Febbraio'=>28,
        'Marzo'=>31,'Aprile'=>30,
        'Maggio'=>31,'Giugno'=>30,
        'Luglio'=>31,'Agosto'=>31,
        'Settembre'=>30,'Ottobre'=>31,
        'Novembre'=>30,'Dicembre'=>31 );

        // stampa gli elementi e le chiavi dell'array
        foreach($mesi as $nomemese => $giorni ) {
           echo "$nomemese ha $giorni giorni";
        }
?>
```

Funzioni su array

print_r($var): stampa il contenuto di una variabile in formato leggibile. Se applicata su un array, il formato di stampa è leggibile ma non molto pratico.
sort ($vet): ordina un array.
rsort ($vet): ordina un array in ordine decrescente.
asort ($vet): ordina un array e mantiene le associazioni degli indici.
arsort ($vet): ordina un array in ordine decrescente e mantiene le associazioni degli indici.
count ($vet): conta gli elementi di un array.
implode("separatore",$array): crea una stringa a partire dagli elementi di un array frapponendo fra i vari elementi il carattere separatore.
explode("separatore", $stringa): crea un array a partire da una stringa composta da elementi suddivisi da "separatore".

Variabili con valori anomali

Valori considerati *false*: PHP interpreta come *false* le variabili con valore:

- 0 o stringa ‘0’
- NULL
- un array vuoto
- una stringa vuota

Valori numerici impliciti: durante le operazioni di calcolo, le stringhe che contengono solo numeri o che iniziano con numeri vengono valutate con il relativo valore numerico. Se non sono convertibili vengono valutate come 0.

```
<?php
$a = “44gatti” % ”3 file”; // $a vale 2
$b = “15galli” + “uova e 5 galline”; // $b vale 15
// perché la seconda stringa è valutata 0
?>
```

Funzioni su stringhe

strlen($var): restituisce la lunghezza di una stringa
trim($str): rimuove gli spazi (spazi, CR, LF, TAB) all'inizio e alla fine di *$str*.
ltrim($str): rimuove gli spazi presenti all'inizio di *$str*
rtrim($str): rimuove gli spazi presenti alla fine di una stringa
substr($str, $inizio [, $num_caratteri]): estrae una stringa da *$str* in base ai parametri passati
str_replace($strsearch, $strreplace, $strsubject): sostituisce tutte le occorrenze di $*strsearch* presenti in $*strsubject* con il contenuto di $*strreplace*
strrev($str): scrive *$str* in ordine inverso
strtolower($str) / strtoupper($str): converte $str in caratteri minuscoli / maiuscoli
ucfirst($str): rende maiuscolo il primo carattere di $str
ucwords($str): rende maiuscolo il primo carattere di ogni parola in $str

Funzioni speciali su variabili

empty($valore): restituisce *true* nel caso in cui la variabile sia vuota (cioè *null*, 0, stringa vuota o non definita), *false* in caso contrario
isset($valore): restituisce *true* se la variabile è definita (not *null*)
is_null($valore): restituisce *true* se la variabile è *null* o non definita
is_int($valore), is_integer($valore): restituiscono *true* se la variabile è numerica intera
is_long($valore), is_float($valore), is_double($valore), is_real($valore): restituiscono *true* se la variabile è numerica decimale
is_string($valore): restituisce *true* se la variabile è stringa
is_array($valore): restituisce *true* se la variabile è un array
is_numeric($valore): restituisce *true* se la variabile può essere considerata di tipo numerico (anche “5” rende true)

Funzioni matematiche

abs(x): restituisce il valore assoluto.
cos(x): restituisce il coseno
sin(x): restituisce il seno
tan(x): restituisce la tangente
sqrt(x): restituisce la radice quadrata
min() e max(): calcolano il massimo e il minimo di una sequenza di valori
log(x): calcola il logaritmo in base e
log10(x): calcola il logaritmo base 10
exp(x): calcola e^x
pow(x,y): calcola x^y
floor(x): arrotonda al più grande intero inferiore a x
ceil(x): arrotonda al più piccolo intero superiore a x
round(x): arrotonda all'intero più vicino
rand(x, y): restituisce un numero casuale compreso tra x e y(opzionali)

Come scrivere il codice

Commenti

- *utili per rileggere il codice a distanza di tempo*
- *utili in caso di lavoro in team*
- *utilizzo eccessivo può essere controproducente*

Nomi di variabili e funzioni

- *Utilizzare nomi significativi*
- *evitare nomi eccessivamente lunghi*

  ex: $vettoredinumeriinteripositivi=array(10,11,12);

  può essere sostituito con $vet=array(10,11,12);

Organizzazione del codice

- *Suddividere il codice in blocchi logici di istruzioni omogenee organizzandolo in **funzioni** e **file** di codice sorgente distinti*

Come scrivere il codice - 2

*L'**indentazione** è una tecnica di scrittura del codice sorgente dei programmi che ha lo scopo di aumentarne la leggibilità [...]. È estremamente diffusa e il suo utilizzo viene considerato come una norma fondamentale di buona programmazione.*

*Definizione tratta da **it.wikipedia.org***

Si basa sull'utilizzo degli spazi bianchi per suddividere visivamente il codice in blocchi di istruzioni omogenee legate tra loro, come quelle appartenenti a cicli, flussi di sequenza di istruzioni condizionali, funzioni, etc.

Un tutorial sull'argomento è presente all'indirizzo:
http://php.html.it/articoli/leggi/889/scrivere-codice-pulito-e-ben-formattato/

Codice indentato e non indentato

```php
<?php
   // codice indentato
   $a = 1;
   switch($a){
     case 0:
       echo "Zero";
       break;
     case 1:
       echo "Due";
       for ($i = 0;$i <= 10;$i++) {
           $ris = 2 * $i;
           echo $ris;
       }
       break;
     default:
       print "altro";
       break;
   }
?>
```

```php
<?php
// codice non indentato
$a = 1;
switch($a){
case 0:
echo "Zero";
break;
case 1:
echo "Due";
for ($i = 0;$i <= 10;$i++) {
$ris = 2 * $i;
echo $ris;
}
break;
default:
print "altro";
break;
}
?>
```

Codice distribuito su più file

E' possibile ripartire il codice sorgente dello script su più file, utilizzando le funzioni *include()* e *require()*.

- Le istruzioni *include()* e *require()* includono e valutano il contenuto del file specificato.
- I due costrutti sono identici in ogni aspetto eccetto per il trattamento degli errori. Il primo produce un "*Warning*" mentre l'altro restituisce un "*Fatal Error*" e blocca l'esecuzione dello script.
- L'inclusione avviene nel momento in cui l'istruzione di inclusione viene eseguita (se viene eseguita).
- Una volta incluso, il blocco di codice definito nel file esterno, viene considerato dall'interprete PHP come se fosse stato definito nel file chiamante.

Esempi – include() e require()

inclusi/fioraio.php

```
<?php
$nome = “Mario”;
$cognome = “Rossi”;
$esiste_fioraio = true;
?>
```

inclusi/variabili.php

```
<?php
$fiore = “rododendro”;
$colore = “rosso”;
?>
```

stampa.php

```
<?php
$dati_fioraio = “./inclusi/fioraio.php”;
include($dati_fioraio);
if($esiste_fioraio) include(“./inclusi/variabili.php”);
echo “$nome $cognome possiede un $fiore $colore”;
?>
```

Visibilità di una variabile

Si definisce come *visibilità* (o *portata* o *scope*) di una variabile il contesto in cui la variabile risulta definita.

- All'interno di uno script, una variabile è solitamente visibile solo dalle istruzioni successive all'istruzione in cui la variabile viene utilizzata per la prima volta (definizione implicita).
- Una variabile definita come variabile locale di una funzione è visibile solo all'interno di tale funzione.
- Una variabile utilizzata come parametro di ingresso di una funzione è visibile solo all'interno di tale funzione.
- All'interno di funzioni sono visibili solo variabili locali o parametri ma non variabili definite fuori dalla funzione.
- Tramite l'istruzione *global* è possibile modificare la visibilità delle variabili.

Esempi - Visibilità di variabili

**Dentro la funzione Prova() le variabili definite all'esterno non risultano visibili. La variabile $a all'interno della funzione è diversa da quella definita all'esterno.
Fuori dalla funzione le variabili $b e $e non sono definite**

```
<?php
$a = 1;
function Prova($b){
$e = 11;
echo $a; // La variabile $a non è definita nella funzione
echo $b; // La variabile $b invece si
$a = 5;  // Definisco la variabile $a che è diversa da
echo $a; // quella definita all'esterno della funzione
$b++;
}
$c = 5;
Prova($c);
echo $a;        // Stampa il valore 1
echo “$b $e”;   /* Non stampa niente: le variabili $b e $e
           fuori dalla funzione non risultano definite */
echo $c;        // La variabile $c non è stata modificata
?>
```

Esempi – Utilizzo di *global*

L'istruzione global rende visibili le variabili $b e $c anche all'interno della funzione. Inoltre la variabile $e che è definita come global risulta visibile anche all'esterno.

```
<?php
$a = 1; $b = 2; $c = 4;
function Prova($b,$f) {
     echo $b; // E' quella locale
     $d = 5;
     global $b, $c, $e;
     $e = 3;
     echo $a; // non è definita
     echo $b; // E' quella globale
     echo $c;
     echo $g;
}

Prova($a,5);
echo $e;
global $g;
$g = 55;
?>
```

Altre funzioni su array

array_search ($src, $array [, $strict]): Ricerca il valore di *$src* in *$array* e ne restituisce la chiave corrispondente. Se la ricerca non ha successo restituisce FALSE. Se impostato a TRUE il parametro *$strict* impone l'identità di tipo, oltre l'uguaglianza di valori tra *$src* e i valori di *$array*.
in_array($search, $array [, $strict]): come il precedente, ma restituisce un valore booleano (TRUE per trovato, FALSE per non trovato).
array_key_exists($chiave, $array): Controlla se l'indice (o chiave) *$chiave* è presente in *$array* e restituisce TRUE in caso di successo o FALSE altrimenti.
array_keys($array): Restituisce un array contenente tutte le chiavi di $array.
array_values($array): Restituisce un array contenente tutti i valori di $array e li indicizza numericamente.
array_reverse($array): Restituisce $array con gli elementi disposti in ordine invertito.

Variabili predefinite - 1

Esistono array associativi che contengono variabili definite in particolari contesti per utilizzi specifici. Le chiavi dell'array sono rappresentate dai nomi delle variabili alle quali si vuole accedere.

$GLOBALS: riferimenti a variabili definite come globali tramite l'istruzione *global*.

$_SERVER: riferimenti a variabili relative all'ambiente di esecuzione del server web e da questo direttamente gestite.

$_ENV: riferimenti a variabili fornite allo script dal sistema operativo ospite.

Variabili predefinite - 2

$_GET: riferimenti a variabili fornite attraverso l'URL (metodo GET del protocollo HTTP).

$_POST: riferimenti a variabili fornite attraverso il metodo POST del protocollo HTTP.

$_COOKIE: riferimenti a variabili fornite allo script attraverso cookies del protocollo HTTP.

$_FILES: riferimenti a variabili fornite allo script attraverso upload di file occorsi col metodo POST HTTP.

$_REQUEST: riferimenti a variabili contenute negli array $_GET, $_POST, $_COOKIE.

$_SESSION: riferimenti alle variabili di sessione dello script. Le variabili di sessione sono tutte quelle variabili definite e visibili durante tutta la durata della navigazione dell'utente.

Esempi - Variabili predefinite

http://nomeserver/script.php?val1=12&val2=ciao

```
<?php
echo $_GET['val1']; // stampa 12
echo “<br>”;
echo $_GET['val2']; // stampa ciao
?>
```

http://nomeserver/script.php?val=Ciao a tutti

```
<?php
// esempio con stringa con spazi
echo $_GET['val']; // stampa Ciao a tutti
?>
```

Gestire dati nei form

Con PHP è possibile gestire e manipolare dati inseriti tramite form. Il passaggio dei dati inseriti è affidato al metodo POST del protocollo HTTP ed è possibile modificare i dati del form o trasferire i valori delle variabili ad un'altra pagina.

```
<form action="pag_dest.php" method="post">
...
...
<input type="submit" value="somma">
</form>
```

action = La pagina di destinazione (dove i dati saranno elaborati)
method = Il metodo di passaggio dei dati (solitamente POST)

Esempio: gestire dati nei form - 1

I dati vengono trasferiti dal modulo contenuto in form1.html alla pagina di elaborazione vedi.php tramite il metodo POST del protocollo HTTP

form1.html

```
<form action="vedi.php" target="_blank"
method="post">
<input type="text" name="dato" size=20 maxlength=20>
<input type="submit" value="Conferma">
</form>
```

vedi.php

```
<?php
echo “Il valore impostato nel form è ”;
echo $_POST['dato'];
?>
```

Esempio: gestire dati nei form - 2

```
<?php
function somma($uno, $due) {
     return $uno + $due;
}

$a = $_POST['val1']; // valore impostato nel form
$b = $_POST['val2']; // valore impostato nel form
$res = ($a or $b) ? somma($a,$b) : null;
?>
<form action="form.php" method="post">
<input type="text" name="val1" size=6 maxlength=6
     value="<? echo $val1;?>"> Primo addendo<br>
<input type="text" name="val2" size=6 maxlength=6
     value="<? echo $val2;?>"> Secondo addendo<br>
<br>
<input type="text" name="res" size=8 maxlength=8
value="<? echo $res;?>"> Risultato
<input type="submit" value="Somma">
</form>
```

Esercizio form

Realizzare un pagina con i seguenti input in un form:

- Nome
- Cognome
- Data di nascita

Realizzare un file con 2 funzioni:

- calcolo età
- giorni mancanti al prossimo compleanno

Al submit del form devono essere visualizzate le seguenti informazioni:
Ciao Nome Cognome,
Hai XX anni e mancano YYY al tuo compleanno.
Se la persona non è maggiorenne visualizzare anche:
Non hai 18 anni, spiacenti non puoi procedere.