Linguaggi di Programmazione per il Web – Parte 2

PHP – Hypertext Preprocessor

Strutture di controllo
e
principali funzioni built-in

Autore

Prof. **Rio Chierego**
riochierego@libero.it

Siti Utili

http://www.riochierego.it/mobile

http://www.html.it/

http://www.mrwebmaster.it

http://www.php.net/

http://php.html.it/

La funzione print

La funzione ***print*** (come la funzione ***echo***) può mandare in output qualunque tipo di dato.

```
<?php
echo "Ciao mondo!";
//print funziona allo stesso modo
print "Ciao mondo!";
echo 123;
print 123;
$pippo = "Ciao mondo!";
echo $pippo;
print $pippo;
//possono essere scritti anche cosi
echo ("Ciao mondo!");
print ("Ciao mondo!");
?>
```

(Vedi **PPT-2-Esempio-1.php**)

Sostituzione di variabili

Il PHP riesce a riconoscere (in fase di stampa) le variabile inserite all'interno delle stringhe e lo fa grazie al carattere dollaro ***$***.

```
<?php
$numero = 45;
$veicolo="autobus";
echo "Questo $veicolo ha una capacita di $numero passeggeri";
/*stamperà: Questo autobus ha una capacità di 45 passeggeri*/
?>
```

Se ho la necessità che dopo la variabile non ci sia uno spazio devo racchiudere la variabile tra ***{ }***

```
<?php
$memoria = 256;
$messaggio="il mio computer ha $memoriaMByte di ram";
/*siccome non esiste nessuna variabile con il nome $memoriaMByte assegnarà a $messaggio: il mio computer ha di ram*/
$messaggio="il mio computer ha {$memoria}MByte di ram";
//assegnerà a messaggio: il mio computer ha 256MByte di ram
?>
```

(Vedi **PPT-2-Esempio-2.php**)

ATTENZIONE: questo discorso vale solo per stringhe racchiuse tra virgolette e non tra apici

Conversioni tra i tipi: la funzione settype

E' possibile forzare conversione di una variabile da un tipo all'altro utilizzando le funzioni dette di "cast"

```
<?php
$pippo = (int) $var;
$pluto = (boolean) $var;
$paperino = (float) $var;
$topolino = (string) $var;
?>
```

Le funzioni precedenti **non** trasformano il tipo di $var in modo permanente ma solo per l'operazione corrente.
Se si vuole cambiare in modo permanente il il tipo di una variabile occorre utilizzare la funzione ***settype***

```
<?php
$var = 39;
settype ($var,"string");
?>
```

Debug: le funzioni var_dump e print_r

Per tenere sotto controllo quello che succede in uno script ed individuare eventuali errori esistono alcune funzioni apposite per il **debug**:

```
<?php
//Per controllare il valore di una variabile:
$pluto = "un topolino" + 39;
 var_dump ($pluto);
//verrà stampato: int(39)
?>
```

La funzione ***print_r*** è utile per stampare array in modo strutturato:

```
<?php
$prova = array("blu","rosso");
print_r ($prova);
/*verrà stampato:
Array
(
  [0] => blu
  [1] => rosso
)
*/
?>
```

(vedi **PPT-2-Esempio-3.php**)

Assegnazione per riferimento

Quando si effettua un assegnazione il valore di una variabile viene copiato in un'altra:

```
<?php
$a = 5;
$b = $a;
/*Abbiamo 2 variabili (a e b) con lo stesso valore (5) */
?>
```

Utilizzando l'operatore **&** è possibile invece fare in modo che due variabili puntino alla stessa casella di memoria:

```
<?php
$a = 5;
$b = &$a;
//se modifichiamo $a cambia anche $b
$a = $a+1;
echo $b; //stamperà 6
?>
```

I metodi GET e POST

Per poter rendere **DINAMICO** un sito web grazie al linguaggio PHP è necessario che sia possibile passare dei valori in ingresso ai nostri script.

Il **metodo GET** consiste nell'accodare i dati all'indirizzo della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome/valore dei dati che ci interessano. Nome e valore sono separati da un segno di uguale. Le diverse coppie nome/valore sono separate dal segno '&'.

<a href="prodotto.php?par1=a7&par2=25" target=_blank>

I metodi GET e POST

La stringa che si trova dopo il punto interrogativo, contenente nomi e valori dei parametri, viene detta **query string**.

Quando la pagina **prodotto.php** viene richiamata in questo modo, essa avrà a disposizione, al suo interno, un particolare array globale ***$_GET*** dove **le chiavi sono i nomi dei parametri e i valori corrispondono ai valori dei paramentri.**

$_GET['par1'] conterrà 'a7'
$_GET['par2'] conterrà 25 (vedi **PPT-2-Esempio-4.php**)

I metodi GET e POST

Il **metodo POST** viene utilizzato con i **moduli (form)**: quando una pagina HTML contiene un **tag <form>,** uno dei suoi attributi è "method", che può valere **GET** o **POST**.

Se il metodo è **GET**, i dati vengono passati nella query string, come abbiamo visto prima. I dati che vengono passati attraverso il metodo GET sono memorizzati nell'array globale **$_GET**

Se il metodo è **POST**, i dati vengono invece inviati in maniera da non essere direttamente visibili per l'utente, attraverso la richiesta HTTP che il browser invia al server.

I dati che vengono passati attraverso il **metodo POST** sono memorizzati nell'array **$_POST**. Questo array si comporta esattamente come **$_GET**:

```
<FORM action="PPT-2-esempio-5.php" method="POST">
<input type="text" name="nome">
<input type="submit" name="submit" value="invia">
</FORM>
```

(vedi **PPT-2-esempio-5.htm**)

Avremo:

$_POST['nome'] che conterrà il valore che l'utente ci scriverà dentro

(vedi **PPT-2-esempio-5.php**)

Strutture di controllo: selezione unaria (IF)

Dopo l'**if**, deve essere indicata fra parentesi **un'espressione booleana** da valutare (condizione). Dunque, se la condizione è vera, l'istruzione successiva viene eseguita. In caso contrario, viene ignorata.

Potremmo anche avere la necessità di eseguire, nel caso in cui la condizione sia vera, non una sola ma più istruzioni. Dobbiamo ricordarci di comprendere questo blocco di istruzioni fra due parentesi graffe:

```
if ( <condizione_booleana> )
  {
  <blocco di istruzioni ramo allora – condizione booleana TRUE >
  }
```

Dobbiamo notare un particolare: per la prima volta, vediamo che in questo costrutto mancano i punto e virgola. Infatti la 'if' e la condizione espressa fra parentesi non devono averli.

Strutture di controllo: selezione binaria (IF…ELSE)

```
if ( <condizione_booleana> )
   {
   <blocco di istruzioni ramo ALLORA – condizione booleana è TRUE >
   }

else
   {
   <blocco di istruzioni ramo ALTRIMENTI – condizione booleana è FALSE>
   }
```

La parola chiave **'else'**, che significa 'altrimenti', deve essere posizionata subito dopo la parentesi graffa di chiusura del codice previsto per il caso 'vero'. Anche per 'else' valgono le stesse regole: niente punto e virgola, parentesi graffe obbligatorie se dobbiamo esprimere più di un'istruzione, altrimenti facoltative.

Le istruzioni 'if' ed 'if…else' possono essere nidificate una dentro l'altra.

Strutture di controllo: selezione binaria (IF…ELSE)

Esercizio 1: Creare una form che permette di inserire un numero e al sumbit richiamare un PHP che stampi a video se il numero è pari o dispari.

Esercizio 2: Creare una form che permette di inserire 2 numeri e al submit richiamare un PHP che stampi a video se il primo numero è divisibile per il secondo e viceversa

Esercizio 3: Creare una form che permette di inserire 2 numeri e al submit richiamare un PHP che stampi a video il maggiore dei 2

Esercizio 4: Creare una form che permette di inserire 3 numeri e al submit richiamare un PHP che li stampa a video in ordine crescente

Strutture di controllo: l'operatore ternario

L'operatore ternario è così chiamato perchè è formato da tre espressioni: il valore restituito è quello della seconda o della terza di queste espressioni, a seconda che la prima sia vera o falsa. In pratica, si può considerare, in certi casi, una maniera molto sintetica di effettuare un **'if...else'**

```
$aggettivo = ($altezza >= 180) ? 'alto' : 'normale' ;
```

La variabile **$aggettivo** prenderà il valore 'alto' se la variabile $altezza è maggiore o uguale a 180, e 'normale' nel caso opposto. Come vediamo, l'espressione condizionale è contenuta fra parentesi tonde ed è seguita da un punto interrogativo, mentre due punti separano la seconda espressione dalla terza.

Questo costrutto può essere utilizzato, ad esempio, per valorizzare velocemente una variabile senza ricorrere all'if:

```
if ($altezza >= 180)
    $tipologia = 'alto' ;
else
    $tipologia = 'normale';
```

(vedi **PPT-2-esempio-6.php**)

Strutture di controllo: selezione ennaria (SWITCH)

```
switch ( <nome_variabile> | <valore-espressione> )
  {
  case <valore_1> :
     {
     <blocco istruzioni se valore_1>
     break;
     }
  case <valore_2> :
     {
     <blocco istruzioni se valore_2>
     break;
     }
  …..
 default:
     {
    <blocco istruzioni se non è nessuno dei valori precedenti>
     }
  }
```

N.B. I valori considerati **<valore_1>, <valore_2>**,…,<valore_n> devono essere coerenti con il tipo di <nome_variabile> oppure **<valore_espressione>**

Strutture di controllo: selezione ennaria (SWITCH)

L'istruzione 'switch' prevede che indichiamo, fra parentesi, un'espressione o una variabile che verrà valutata per il suo valore.

Di seguito, tra parentesi graffe, esprimeremo una serie di espressioni da confrontare con quella indicata prima: dal momento in cui ne trova una il cui valore è uguale, PHP esegue il codice indicato di seguito, **fino a quando non incontra un'istruzione 'break'**.

L'istruzione 'default' può essere indicata come 'ultimo caso', che si considera verificato quando nessuno dei casi precedenti è risultato vero.

L'indicazione 'default' può anche essere assente, ma quando c'è deve essere l'ultima dello switch.

Strutture di controllo: cicli precondizionali (WHILE)

Il ciclo 'while' prevede che alla parola chiave 'while' segua, fra parentesi, la condizione da valutare, e fra parentesi graffe, il codice da rieseguire fino a quando **tale condizione rimane vera (TRUE)**.

```
while ( <condizione_logica> )
  {
  <blocco istruzioni da eseguire in ciclo>
  }
```

(vedi **PPT-2-esempio-7.php**)

Il concetto fondamentale è che l'esecuzione del ciclo termina quando la condizione fra parentesi **non è più verificata ossia diventa falsa (FALSE)**

Strutture di controllo: cicli postcondizionali (DO…WHILE)

Esiste un'altra forma, simile al 'while', con la quale possiamo assicurarci che il codice indicato tra le parentesi graffe venga eseguito almeno una volta fino a quando **la condizione rimane vera (TRUE):** si tratta del 'do...while'

```
do
  {
  <blocco istruzioni da eseguire in ciclo>
  }
while ( <condizione_logica> ) ;
```

Il 'while' viene spostato **dopo** il codice da ripetere, ad indicare che la valutazione della condizione viene eseguita solo dopo l'esecuzione del codice fra parentesi graffe.

Il concetto fondamentale è che l'esecuzione del ciclo termina quando la condizione fra parentesi **non è più verificata ossia diventa falsa (FALSE)**

Strutture di controllo: cicli precondizionali (FOR)

Iniziamo subito con un esempio: supponiamo di voler mostrare **i multipli da 1 a 10 di un numero**, ad esempio **5**

```
for ( $i = 1; $i <= 10; $i++ )
  {
  $ris = 5 * $i;
  echo $ris." ";
  }
```

```
for ( $i = 10; $i >= 1; $i-- )
  {
  $ris = 5 * $i;
  echo $ris." "
  }
```

Questi due costrutti utilizzano la parola chiave **'for'**, seguita, fra parentesi, dalle istruzioni per definire il ciclo; di seguito, si racchiudono fra parentesi graffe tutte le istruzioni che devono essere eseguite ripetutamente.

Le tre istruzioni inserite fra le parentesi tonde vengono trattate in questo modo:
la prima viene eseguita una sola volta, all'inizio del ciclo;
la terza viene eseguita alla fine di **ogni** iterazione del ciclo;
la seconda deve essere una condizione, e viene valutata **prima** di ogni iterazione del ciclo: quando risulta falsa, l'esecuzione del ciclo viene interrotta. Quando invece è vera, le istruzioni fra parentesi graffe vengono eseguite.

Strutture di controllo: cicli (CONTINUE - BREAK)

Uscire da un ciclo. PHP termina l'esecuzione di un ciclo quando la condizione a cui è sottoposto non è più verificata.

Possiamo però dire a PHP di non completare la presente iterazione e passare alla successiva (con '***continue***') oppure di interrompere definitivamente l'esecuzione del ciclo (con '***break***'). Vediamo un esempio:

```
for ($i = 1; $i <= 50; $i++)
   {
   if ($i % 10 == 0)
       {
       break;
       }
  elseif ($i % 3 == 0)
      {
      continue;
      }
   echo 'valore: '.$i.'<br>';
   }
```

(vedi **PPT-2-esempio-8.php**)

Il ciclo si ripeterà solo 10 volte anziché 50 e i valori multipli di 3 non verranno stampati

Strutture di controllo: Esercizi selezione

Esercizio 1: Creare una form che permette di inserire 2 numeri e che permette di scegliere con dei checkbutton (o option button) (+,-,*,/,%) l'operazione da compiere e al sumbit richiamare un PHP che stampi a video il risultato dell'operazione (CALCOLATRICE)

Esercizio 2: Aggiungere all'esercizio precedente il controlli per evitare di dividere per 0 stampando in quel caso un messaggio di errore personalizzato. Aggiungere anche il controllo che i due numeri siano stati effettivamente inseriti.

Strutture di controllo: Esercizi cicli

Esercizio 3: Creare una form che permette di inserire un numero e al submit richiamare un PHP che stampi a video (in una tabella HTML) la tabellina (dall'1 al 10) del numero dato in input. (fare l'esercizio usando i diversi modi di gestire i cicli)

Esercizio 4: modificare l'esercizio precedente in modo che sia possibile scegliere il numero di partenza e il numero di arrivo della tabellina

Gli array

Esempio di array:

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

In questo modo, PHP associa a ciascuno dei valori che abbiamo elencato un **indice numerico**, a partire da **0**. Per riferirsi ad un singolo elemento dell'array si indica il nome dell'array seguito dall'indice contenuto fra parentesi quadre:

Esiste poi un metodo per aggiungere un valore all'array; questo metodo può essere usato anche, come alternativa al precedente, per definire l'array:

$colori[ ] = 'blu';

Con questo codice verrà creato un nuovo elemento nell'array **$colori**, che avrà l'indice **5**. Questa sintassi è valida anche per definire un array: infatti, se l'array **$colori** non fosse ancora definito, questa istruzione lo avrebbe definito creando l'elemento con indice **0**.

$colori[3] = 'arancio';

$colori[7] = 'viola';

L'elemento con indice **3**, che prima valeva **'verde'**, avrà il valore cambiato in **'arancio'**. Inoltre avremo un nuovo elemento, con indice **7**, con il valore **'viola'**.

Dopo queste istruzioni, l'array ha un "buco", perchè dal codice 5 si salta direttamente al codice 7: se useremo di nuovo l'istruzione di "incremento" con le parentesi quadre vuote, il nuovo elemento prenderà l'indice 8.

Gli array

Gli indici degli elementi non sono necessariamente numerici. Possono essere anche delle stringhe e le chiavi numeriche e stringa possono coesistere nello stesso array.

```
$formazione[1] ='Buffon';
$formazione[2] ='Panucci';
$formazione[3] ='Nesta';
….
$formazione[11] ='Vieri';
$formazione[ 'ct' ] = 'Trapattoni';
```

Abbiamo creato un array con dodici elementi, di cui undici con chiavi numeriche, ed uno con chiave associativa. Se in seguito volessimo aggiungere un elemento usando le parentesi quadre vuote, il nuovo elemento prenderà l'indice 12.

Avremmo potuto creare lo stesso array usando l'istruzione di dichiarazione dell'array, così:

```
$formazione = array(1 => 'Buffon', 'Panucci', 'Nesta', …, 'Vieri', 'ct' => 'Trapattoni');
```

(vedi **PPT-2-esempio-9.php)**

Gli array

Alcuni esempi:

```
$persona[ 'nome' ] = 'Mario';          //corretto
$persona[cognome] = 'Rossi';           //funziona, ma genera un errore 'notice'

echo $persona[ 'cognome' ];            //stampa 'Rossi': corretto
echo "ciao $persona[ 'nome' ]";        //NON FUNZIONA, GENERA ERRORE

echo "ciao " . $persona[ 'nome' ];
//corretto: come alternativa, usiamo il punto per concatenare

echo "ciao {$persona[ 'nome' ]} ";
//corretto: usiamo le graffe
```

Gli array a più dimensioni

Un array a più dimensioni è un array nel quale uno o più elementi sono degli array a loro volta. Supponiamo di voler raccogliere in un array i dati anagrafici di più persone: per ogni persona registreremo nome, cognome

```
$persone = array(
                    array('nome' => 'Mario', 'cognome' => 'Rossi'),
                    array('nome' => 'Paolo', 'cognome' => 'Bianchi'),
                    array('nome' => 'Luca', 'cognome' => 'Verdi')
                    );
echo $persone[0]['cognome'];       // stampa 'Rossi'
echo $persone[2]['nome'];           // stampa 'Luca'
```

Vediamo un altro esempio:

```
$persone = array(
              1 => array('nome' => 'Mario Rossi', 'residenza' => 'Roma', 'ruolo' => 'impiegato'),
              2 => array('nome' => 'Paolo Verdi',  'residenza' => 'Torino', 'ruolo' => 'operaio'),
              'totale_elementi' => 2
               );
echo $persone[1]['residenza'];        // stampa 'Roma'
echo $persone['totale_elementi'];    // stampa '2'
```

(vedi **PPT-2-esempio-10.php**)

Gli array e la struttura di controllo FOREACH

Esiste un ultimo tipo di ciclo, ed è un ciclo particolare perché é costruito appositamente per il trattamento degli array: si tratta del 'foreach'. Questo ci permette di costruire un ciclo che viene ripetuto per ogni elemento dell'array che gli passiamo. La sintassi è la seguente:

```
foreach ($array as $chiave => $valore)
  {
  <blocco di istruzioni>
  }
```

All'interno delle parentesi graffe avremo a disposizione, nelle due variabili che abbiamo definito **$chiave** e **$valore**, l'indice e il contenuto dell'elemento che stiamo trattando. E' da notare che, nel caso ci interessi soltanto il valore e non la chiave (ad esempio perché le chiavi sono numeriche), possiamo anche evitare di estrarre la chiave stessa:

```
foreach ($arr as $valore)
  {
  <blocco di istruzioni>
  }
```

Gli Array

Esercizio 1: Creare un form che permetta di inserire 10 numeri. Creare uno script PHP per memorizzare i 10 numeri in un array e che mostri il Maggiore, il Minore e la Media tra i 10 numeri.

Esercizio 2: Dato un array iniziale che contiene 10 numeri (ad esempio: array(45,56,32,900,76,5,333,4,1,20) **) creare un form che chieda al navigatore di indovinare quali sono i 10 numeri.**

Creare lo script PHP che verifica i numeri inseriti e che stampi a video quali e quanti numeri sono stati indovinati.
DIAMO PER SCONTATO CHE I NUMERI SIANO TUTTI DIVERSI

Esercizio

Proviamo a costruire una semplice calcolatrice:

Step 1: create una pagina chiamata **calcola.php** e date due variabili **$a** e **$b** scrivete il codice che ne faccia la somma,la sottrazione, la moltiplicazione, la divisione e il modulo

Step 2: modificate il codice in modo che le variabili **$a** e **$b** arrivino come parametri in **GET**

Step 3: create una altra pagina chiamata **index.php** dove tramite un form permettete di inviare i valori per le variabili **$a** e **$b** alla pagina **calcola.php**

Step 4: modificate la pagina **calcola.php** in modo che l'operazione da eseguire venga stabilita in base al valore di un parametro in **GET**

Step 5: modificate la pagina **index.php** in modo che ci siano dei **check button** (oppure **option button**)per scegliere l'operazione da fare

Step 6: aggiungente controlli per assicurare l'assenza di errori (divisione per 0, verifica dei tipi di variabili, verifica dei campi obbligatori ecc…)