Linguaggi di Programmazione per il Web – Parte 3

PHP – Hypertext Preprocessor

Funzioni built-in
e
Funzioni utente

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/

Le funzioni

Una funzione è un insieme di istruzioni che hanno lo scopo di eseguire determinate operazioni. Le funzioni ci consentono di non dover riscrivere il codice ogni volta che abbiamo la necessità di rieseguire quelle operazioni: ci basta infatti richiamare l'apposita funzione, fornendole i **parametri** di cui ha bisogno per la sua esecuzione.

Ciò che caratterizza le funzioni, è il fatto di richiedere parametri (quali e quanti), nonché quello di restituire o meno un valore.

La sintassi con la quale si richiama una funzione, cioé si chiede a PHP di eseguirla, è molto semplice:

nome_funzione ( );

Si tratta semplicemente di indicare il nome della funzione, **seguito da parentesi tonde,** che devono contenere i parametri da passare alla funzione, ma vanno **obbligatoriamente** indicate anche se non ci sono parametri. Nel caso poi in cui la funzione restituisca un valore, possiamo indicare la variabile che utilizzerà questo valore:

$valore = nome_funzione ( );

Le funzioni in PHP

Le **funzioni** possono essere incorporate nel linguaggio **(built-in) oppure definite dall'utente**. Esistono moltissime funzioni incorporate in PHP che possono essere utilizzate per gli usi più disparati.

E' ovviamente impossibile conoscerle tutte a memoria, diventa quindi importante avere a disposizione un manuale del linguaggio PHP e saperlo leggere.

Il miglior manuale a disposizione è presente all'indirizzo http://www.php.net ed ha una particolarità proprio per le funzioni:
digitando **http://www.php.net/nome_funzione** viene proposta direttamente la pagina relativa del manuale

D'ora in poi quando parleremo di funzioni utilizzeremo la notazione e che viene utilizzata dal sito **www.php.net**

Le principali funzioni built-in in PHP (funzioni sulle variabili)

bool empty (mixed var)

verifica se la variabile che le passiamo è vuota oppure no. Per 'vuota' si intende che la variabile può contenere una stringa vuota '' o un valore numerico pari a 0, ma può anche essere non definita o essere impostata al valore NULL. **Restituisce** un valore booleano (TRUE o FALSE).

bool isset (mixed var [, mixed var [, ...]])

verifica se la variabile è definita. Una variabile risulta non definita quando non è stata inizializzata o è stata impostata col valore NULL. **Restituisce** un valore booleano (TRUE o FALSE).

bool is_null (mixed var)

verifica se la variabile equivale a NULL. **Restituisce** un valore booleano (TRUE o FALSE).

Le principali funzioni built-in in PHP (funzioni sulle variabili)

bool is_int (mixed var)
verifica se la variabile è di tipo intero. **Restituisce** un valore booleano.

bool is_float (mixed var)
verifica se la variabile è di tipo numerico double (o float). **Restituisce** un valore booleano.

bool is_string (mixed var)
verifica se la variabile è di tipo stringa. **Restituisce** un valore booleano.

bool is_array (mixed var)
verifica se la variabile è di tipo array. **Restituisce** un valore booleano.

bool is_numeric (mixed var)
verifica se la variabile contiene un valore numerico. Nel caso di una stringa che contiene valori numerici. restituisce vero. **Restituisce** un valore booleano.

Le principali funzioni built-in in PHP (funzioni sulle variabili)

bool print_r (mixed expression [, bool return])
stampa (direttamente sul browser) informazioni relative al contenuto della variabile che le abbiamo passato. E' utile in fase di **debug**, quando a seguito di comportamenti 'strani' del nostro script vogliamo verificare il contenuto di certi dati. Se il valore passato è un array, la funzione ne evidenzia le chiavi ed i valori relativi. **Restituisce** un valore booleano.

void unset (mixed var [, mixed var [, mixed ...]])
distrugge la variabile specificata. In realtà non si tratta di una funzione, ma di un costrutto del linguaggio, e ne abbiamo già parlato. Dopo l'unset(), l'esecuzione di empty() o is_null() sulla stessa variabile restituirà vero, mentre isset() restituirà falso. **Non restituisce** valori.

Le principali funzioni built-in in PHP (funzioni sulle variabili)

Vediamo ora qualche esempio:

```
$b = empty($a);          //$a non è ancora definita, quindi $b sarà TRUE
$a = 5;
$b = isset($a);          //$b sarà TRUE
$b = is_float($a);       //$b sarà FALSE: $a è un intero
$b = is_string($a); /    //$b sarà FALSE: $a è un intero


$a = '5';
$b = is_int($a);          //$b sarà FALSE: $a ora è una stringa
$b = is_string($a);       //$b sarà TRUE:   $a ora è una stringa
$b = is_numeric($a);      //$b sarà FALSE: $a ora è una stringa

unset($a);                //eliminiamo la variabile $a;
$b = isset($a);           //$b sarà FALSE: $a è stata distrutta
```

Solitamente queste funzioni vengono usate come argomenti di condizioni di selezioni

```
if (empty($a))
  {
  print ('$a è vuota o non definita!');
  }
else
  {
 print ('$a contiene un valore');
  }
```

(vedi **PPT-3-esempio-1.php**)

Le principali funzioni built-in in PHP (funzioni sulle stringhe)

int **strlen** (string str)
verifica la lunghezza della stringa, cioè il numero di caratteri che la compongono. **Restituisce** un numero intero.

string **trim** (string str [, string charlist])
elimina gli spazi all'inizio e alla fine della stringa. **Restituisce** la stringa modificata.

string **ltrim** (string str [, string charlist])
come trim ma solo per l'inizio della stringa. **Restituisce** la stringa modificata.

string **rtrim** (string str [, string charlist])
come trim ma solo per la fine della stringa. **Restituisce** la stringa modificata.

string **ucfirst** (string str)
trasforma in maiuscolo il primo carattere della stringa. **Restituisce** la stringa modificata.

string **ucwords** (string str)
trasforma in maiuscolo il primo carattere di ogni parola della stringa, intendendo come parola una serie di caratteri che segue uno spazio. **Restituisce** la stringa modificata.

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

Le principali funzioni built-in in PHP (funzioni sulle stringhe)

string **substr** (string str, int start [, int length])
restituisce una porzione della stringa, in base al secondo parametro (che indica l'inizio della porzione da estrarre), e all'eventuale terzo parametro, che indica quanti caratteri devono essere estratti. Se il terzo parametro non viene indicato, viene restituita tutta la parte finale della stringa a partire dal carattere indicato.

Da notare che i caratteri **vanno contati a partire da zero**, per cui se si chiama la funzione con **substr(stringa, 4)** verranno restituiti tutti i caratteri **a partire dal quinto**. Si può anche indicare un numero negativo come carattere iniziale: in questo caso, il carattere iniziale della porzione di stringa restituita verrà contato a partire dal fondo.

Ad esempio, con **substr(stringa, -5, 3)** si otterranno tre caratteri a partire dal quintultimo (da notare che in questo caso il conteggio non inizia da zero, ma da 1: cioè -1 indica l'ultimo carattere, -2 il penultimo e così via).

Se infine si indica un numero negativo come terzo parametro, tale parametro non verrà più utilizzato come numero di caratteri restituiti, ma come numero di caratteri **non** restituiti a partire dal fondo. Esempio: **substr(stringa, 3, -2)** restituisce i caratteri dal quarto al terzultimo.

La funzione **restituisce** la porzione di stringa richiesta.

(vedi **PPT-3-esempio-3.php**)

Le principali funzioni built-in in PHP (funzioni sulle stringhe)

mixed **str_replace** (mixed search, mixed replace, mixed subject)
effettua una sostituzione della prima stringa (search) con la seconda (replace) all'interno della terza (subject).

Ad esempio: **str_replace('p', 't', 'pippo')** sostituisce le 'p' con le 't' all'interno di 'pippo', e quindi restituisce 'titto'. Al posto delle stringhe e possibile passare un array di stringhe, l'operazione verrà ripetuta su tutte le voci dell'array. **Restituisce** la terza stringa modificata.
*Esiste anche la funzione **str_ireplace()**, che è equivalente ma che cerca la prima stringa nella terza senza tener conto della differenza fra maiuscole e minuscole.*

int **strpos** (string str1, string str2)
cerca la posizione della seconda stringa (str2) all'interno della prima (str1).

Ad esempio: **strpos('Lorenzo', 're')** restituisce 2. **Restituisce** un intero che rappresenta la posizione **a partire da 0** della stringa cercata. Se la seconda stringa non è presente nella prima, restituisce il valore booleano FALSE. *La funzione **stripos()** fa la stessa ricerca senza tenere conto della differenza fra maiuscole e minuscole.*

(vedi **PPT-3-esempio-4.php**)

Le principali funzioni built-in in PHP (funzioni sulle stringhe)

string **strstr** (string haystack, string needle)
cerca la seconda stringa all'interno della prima, e restituisce la prima stringa a partire dal punto in cui ha trovato la seconda. **strstr('Lorenzo', 're')** restituisce 'renzo'. **Restituisce** una stringa se la ricerca va a buon fine, altrimenti il valore booleano FALSE.
*La funzione **stristr()** funziona allo stesso modo ma non tiene conto della differenza fra maiuscole e minuscole.*

string **strtolower** (string str)
converte tutti i caratteri alfabetici nelle corrispondenti lettere minuscole. **Restituisce** la stringa modificata.

string **strtoupper** (string str)
converte tutti i caratteri alfabetici nelle corrispondenti lettere maiuscole. **Restituisce** la stringa modificata.

(vedi **PPT-3-esempio-5.php**)

Le principali funzioni built-in in PHP (funzioni sugli array)

int **count** (mixed var [, int mode]) o int **sizeof** (mixed var [, int mode])
conta il numero di elementi dell'array. **Restituisce** un intero.

array **array_reverse** (array array [, bool preserve_keys])
inverte l'ordine degli elementi dell'array. Se vogliamo mantenere le chiavi dell'array di input, dobbiamo passare il secondo parametro con valore TRUE. **Restituisce** l'array di input con gli elementi invertiti.

bool **sort** (array array)
ordina gli elementi dell'array. Bisogna fare attenzione, perché questa funzione, contrariamente a molte altre, modifica direttamente l'array che le viene passato in input, che quindi andrà perso nella sua composizione originale. I valori vengono disposti in ordine crescente secondo i contenuti. Le chiavi vanno perse: dopo il sort, l'array avrà chiavi numeriche a partire da 0 secondo il nuovo ordinamento. **Restituisce** un booleano.
La funzione **rsort()** fa la stessa cosa ma effettua l'ordinamento in modo contrario (dal più grande al più piccolo)

(vedi **PPT-3-esempio-6.php**)

Le principali funzioni built-in in PHP (funzioni sugli array)

bool **in_array** (mixed valore, array array)
cerca il valore all'interno dell'array. **Restituisce** un valore booleano: vero o falso a seconda che il valore cercato sia presente o meno nell'array.

bool **array_key_exists** (mixed valore, array search)
cerca il valore fra le **chiavi** (e non fra i valori) dell'array.
Restituisce un valore booleano.

mixed **array_search** (mixed valore, array array)
cerca il valore nell'array e ne indica la chiave. **Restituisce** la chiave del valore trovato o, se la ricerca non va a buon fine, il valore FALSE.

array **array_merge** (array array1, array array2 [, array ...])
fonde gli elementi di due o più array. Gli elementi con chiavi numeriche vengono accodati l'uno all'altro e le chiavi rinumerate. Le chiavi associative invece vengono mantenute, e nel caso vi siano più elementi nei diversi array con le stesse chiavi associative, l'ultimo sovrascrive i precedenti.
Restituisce l'array risultante dalla fusione.

(vedi **PPT-3-esempio-6.php**)

Le principali funzioni built-in in PHP (funzioni sugli array)

mixed **array_pop** (array array)
estrae l'ultimo elemento dell'array, che viene 'accorciato'. **Restituisce** l'elemento in fondo all'array e, contemporaneamente, modifica l'array in input togliendogli lo stesso elemento.

int **array_push** (array array, mixed var [, mixed ...])
accoda i valori indicati all'array. Equivale all'uso dell'istruzione di accodamento $array[]=$valore , con il vantaggio che ci permette di accodare più valori tutti in una volta. **Restituisce** il numero degli elementi dell'array dopo l'accodamento.

mixed **array_shift** (array array)
estrae un elemento come array_pop(), ma in questo caso si tratta del primo. Anche in questo caso l'array viene 'accorciato', ed inoltre gli indici numerici vengono rinumerati. Rimangono invece invariati quelli associativi. **Restituisce** l'elemento estratto dall'array.

int **array_unshift** (array array, mixed var [, mixed ...])
inserisce i valori indicati in testa all'array. **Restituisce** il numero degli elementi dell'array dopo l'inserimento.

(vedi **PPT-3-esempio-7.php**)

Le principali funzioni built-in in PHP (funzioni sugli array)

array **explode** (string separatore, string stringa [, int limit])
trasforma la seconda stringa in un array, usando la prima per separare gli elementi. Il terzo parametro può servire ad indicare il numero massimo di elementi che l'array può contenere (se la suddivisione della stringa portasse ad un numero maggiore, la parte finale della stringa sarà interamente contenuta nell'ultimo elemento).
Ad esempio: **explode(' ', 'ciao Mario')** restituisce un array di due elementi in cui il primo è 'ciao' e il secondo 'Mario'.
Restituisce un array.

string **implode** (string glue, array pieces)
è la funzione opposta di explode(), e serve a riunire in un'unica stringa i valori dell'array. La stringa indicata come primo parametro viene interposta fra tutti gli elementi dell'array. **Restituisce** la stringa risultato dell'aggregazione.

(vedi **PPT-3-esempio-8.php**)

Le principali funzioni built-in in PHP (funzioni sulle date)

Prima di cominciare dobbiamo fare un accenno al **timestamp**, sul quale si basano queste funzioni. Il timestamp è un numero intero, in uso da tempo sui sistemi di tipo UNIX, che **rappresenta il numero di secondi trascorsi a partire dal 1° gennaio 1970.**

Ad esempio, il timestamp relativo alle 15.56.20 del 24 aprile 2003 è 1051192580.

int **time** (void)
è la più semplice di tutte, perché fornisce il timestamp relativo al momento in cui viene eseguita. **Restituisce** un intero (timestamp).

(vedi **PPT-3-esempio-9.php**)

Le principali funzioni built-in in PHP (funzioni sulle date)

string **date** (string format [, int timestamp])
considera il timestamp in input (se non è indicato, prende quello attuale) e fornisce una data formattata secondo le specifiche indicate nel primo parametro. Tali specifiche si basano su una tabella di cui riassumiamo i valori più usati:

Y - anno su 4 cifre
y - anno su 2 cifre
n - mese numerico (1-12)
m - mese numerico su 2 cifre (01-12)
F - mese testuale ('January' - 'December')
M - mese testuale su 3 lettere ('Jan' - 'Dec')
d - giorno del mese su due cifre (01-31)
j - giorno del mese (1-31)
w - giorno della settimana, numerico (0=dom, 6=sab)
l - giorno della settimana, testuale ('Sunday' - 'Saturday')
D - giorno della settimana su 3 lettere ('Sun' - 'Sat')
H - ora su due cifre (00-23)
G - ora (0-23)
i - minuti su due cifre (00-59)
s - secondi su due cifre (00-59)

La funzione **date()** **restituisce** la stringa formattata che rappresenta la data.

Le principali funzioni built-in in PHP (funzioni sulle date)

int **mktime** ([int hh [,int min [,int sec [,int month [,int day [,int year]]]]]])
è una funzione molto utile ma che va maneggiata con molta cura, perché i parametri che richiede in input (tutti numeri interi) hanno un ordine abbastanza particolare, che può portare facilmente ad errori. Sulla base di questi parametri, mktime() calcola il timestamp, ma l'aspetto più interessante è che possiamo utilizzarla per fare calcoli sulle date. Infatti, se ad esempio nel parametro mese passiamo 14, PHP lo interpreterà come 12+2, cioè "febbraio dell'anno successivo", e quindi considererà il mese come febbraio ed aumenterà l'anno di 1. Ovviamente lo stesso tipo di calcoli si può fare su tutti gli altri parametri. **Restituisce** un intero (timestamp).

bool **checkdate** (int month, int day, int year)
verifica se i valori passati costituiscono una data valida. **Restituisce** un valore booleano.

(vedi **PPT-3-esempio-9.php**)

Le funzioni definite dall'utente

L'utilizzo di **funzioni** è fortemente consigliato ogni qual volta si debba ripetere più volte una stesse serie di istruzioni, eventualmente su variabili diverse. Le ragioni sono sicuramente una migliore leggibilità del codice, una maggiore facilità nella manutenzione e una compattezza del codice.

In PHP la dichiarazione di una funzione è del tipo

function *miafunzione* (arg1, arg2, arg3=default3, ...)
{
<blocco di istruzioni >
[return <espr>;]
}

Obbligatorie le parentesi graffe per la delimitazione del blocco di istruzioni che verrà eseguito all'invocazione della nostra funzione. Si noti come ai parametri (opzionali) non sia necessario associare un **tipo** ma sia possibile invece indicare un **valore di default** nel caso non venga specificato il valore del parametro nell'invocazione.

I parametri possono essere di ogni tipo, compresi quindi **array** e **oggetti**.

Le funzioni definite dall'utente

Possiamo prevedere delle funzioni in cui non è obbligatorio che tutti gli argomenti previsti vengano passati al momento della chiamata. Abbiamo visto, infatti, che alcune funzioni di PHP prevedono dei parametri facoltativi.

La stessa cosa vale per le funzioni definite da noi: se infatti, al momento in cui definiamo le funzioni, prevediamo un **valore di default** per un certo parametro, quel parametro diventerà facoltativo in fase di chiamata della funzione, e, nel caso manchi, la funzione utilizzerà il valore di default.

```
function anagrafe($nome, $indirizzo, $cf='non disponibile')
  {
  echo 'Nome: '.$nome.'<br>';
  echo 'Indirizzo: '.$indirizzo.'<br>';
  echo 'Codice fiscale: '.$cf.'<br>';
  return;
  }

anagrafe ('Mario Rossi', 'via Roma 2', 'RSSMRA69S12A944X');
anagrafe ('Paolo Verdi', 'via Parigi 9');
```

(vedi **PPT-3-esempio-10.php**)

Le funzioni definite dall'utente

Per distinguere se i **parametri formali** della funzione devono essere passati **per valore** (questo è il default), cioè la funzione crea internamente una copia della variabile passata e questa può essere modificata senza nessuna ripercussione sulla variabile originale, oppure **per riferimento**, cioè la funzione può modificare la variabile originale, basta anteporre al parametro formale scelto, l'indicazione del simbolo **&.**
In entrambi i casi nella chiamata il simbolo **&** **non dovrà essere indicato**.

```
function quadrato ($a)
  {
  $a = $a * $a;
  return;
  }
```

```
function quadrato1 (&$a)
  {
  $a = $a * $a;
  return;
  }
```

```
……
$b = 2;
……
quadrato ($b);
echo "valore: b=". $b ."<br>";    \* $b passata per PER VALORE verrà mostrato a video 2 *\

quadrato1 ($b);
echo "riferimento: b=". $b;      \* $b passata PER RIFERIMENTO verrà mostrato a video 4 *\
```

(vedi **PPT-3-esempio-12.php**)

by Prof. Rio Chierego

Le funzioni definite dall'utente

All'interno di una funzione riveste particolare importanza l'utilizzo del costrutto

…….
return espr;

dove **espr** può essere un espressione o una semplice variabile.

L'istruzione (opzionale) di **return** permette di *ritornare* uno e **un solo** valore al chiamante, ovviamente quando si necessiti di avere come output della funzione diversi valori è possibile ritornare una variabile di tipo **array** che sarà poi gestita come tale dal chiamante.

Le funzioni definite dall'utente

Vediamo un esempio:

```
function miafunzione ($a, $b)
   {
   $a = $a*10;
   $b = $b*100;
   return array($a, $b);
   }
…….
list ($newa, $newb) = miafunzione (2,3);
…….
```

N.B. list() E' usata per assegnare valori ad una lista di variabili in una sola operazione e funziona solo su array numerici e si aspetta che gli indici numerici partano da 0.

N.B. array() Restituisce un array contenente i parametri. Ai parametri si può dare un indice con l'operatore =>. E' un costrutto del linguaggio usato per rappresentare array letterali, e non una normale funzione

(vedi **PPT-3-esempio-11.php**)

Le funzioni definite dall'utente

La dichiarazione **return** causa la terminazione dell'esecuzione della funzione.

Esempio:

```
function divisione($num1, $num2)
  {
  if ($num2==0)
     {
     return FALSE;
     }
 return ($num1 / $num2);
 }

$a = 5;
$b = 0;
$m = (int) divisione ($a, $b);       //$m diventa 0 = FALSE

$a = 4;
$b = 2;
$m = divisione ($a, $b);             //$m diventa 2
```

(vedi **PPT-3-esempio-13.php**)

by Prof. Rio Chierego

Visibilità (scope) delle variabili

Le variabili utilizzate all'interno di una funzione sono **diverse** da quelle all'esterno. In particolare quelle all'interno sono efficaci solo nei limiti della funzione stessa. Questo è il concetto di **scope** di una variabile, l'ambito di applicazione.

```
function raddoppia ($num1)
  {
 $temp = $num1 * 2;
  }

$variabile = 5;
$raddoppia ($variabile);
echo $temp;
```

Riceveremo un errore perché **$temp** non esiste

Visibilità (scope) delle variabili

Il modo corretto per poter usare un valore locale ad una funzione in altri punti dello script è quello di restituirlo con return

```
function raddoppia($num)
   {
   $valorerestituito = $num * 2;
   return $valorerestituito;
   }

$variabile = 5;
$temp = $raddoppia ($variabile);
echo "Il valore restituito è ".$temp;
```

(vedi **PPT-3-esempio-14.php**)

In questo caso verrà stampato il valore 10
Potevamo utilizzare il nome **$temp** anche all'interno. Tuttavia il **$temp** all'interno della funzione è una variabile differente dal $temp all'esterno

Questa regola non vale per le variabili array **superglobali: $_GET** e **$_POST**

Riuso di funzioni utente con le dichiarazioni include e require

La possibilità di riusare funzioni è una delle caratteristiche preziose del PHP, che fornisce le dichiarazioni ***include*** e ***require*** per poter riusare gli script PHP contenenti dichiarazioni, definizioni di funzioni e anche HTML statico.

funzione.php

```
<?php
function grassetto($stringa)
  {
  return "<B>".$stringa."</B>";
  }
?>
```

(vedi **PPT-3-esempio-15-funzione.php**)

main.php

```
<?php
include "funzioni.php";
grassetto ("questo è grassetto");
?>
```

(vedi **PPT-3-esempio-15-main.php**)

Sia **include** che **require** leggono file esterni la differenza è nel caso di errore.

Se **include** non riesce ad includere un file mostra semplicemente un **warning** mentre **require** blocca l'esecuzione dello script.