

# Introduzione a PHP: Concetti Fondamentali

Davide “*Rocker*” Anastasia

20 dicembre 2005

## Sommario

Dopo alcuni anni di “riflessione”, riprende il mio percorso personale sul PHP. **Questo breve documento parlerà dei principi di PHP, partendo dalla gestione delle variabili per arrivare al controllo di flusso di operazioni (parte pronta, ma non ancora integrata).** Si tratta di una guida base, da leggere velocemente, ma molto importante se ci si avvicina per la prima volta ad un linguaggio di programmazione.

Questo documento è da considerare incompleto e non corretto. Utilizzate pure tutte le informazioni che vi trovate dentro senza problemi, ma sempre con il beneficio del dubbio. Buon lavoro.

## 1. Introduzione

---

### 1.1 Un po’ di storia...

Il PHP nasce a metà del 1994 dalle mani di Rasmus Lerdorf. Egli sviluppò una prima versione mai ufficializzata che utilizzò sulle sue pagine. Il successo arrivò presto e PHP si diffuse grazie a molti fattori. Il più importante fu sicuramente quello di possedere una sintassi C-like, che molto facilitava l’apprendimento del linguaggio ad un gran numero di programmatori già smaliziati con i linguaggi compilati, ed in particolare con il C stesso. PHP 3 fu sicuramente la versione che riscosse il maggior successo, ma il maggior incremento tecnologico si ebbe con la versione 4 e l’entrata nello sviluppo della Zend. PHP da allora è stato continuamente migliorato.

L’attuale versione di PHP è la 5, ma i concetti esposti sono così basilari che mai si farà riferimento alla versione. Quando vi troverete a lavorare con PHP però consultate attentamente la documentazione, che tra le altre cose indica da quando una data funzione è disponibile.

Quando ho incominciato a lavorare con PHP la versione in uso era la 3, anche se già si vociferava di una versione 4, uscita infatti dopo poco. Si parla del 2000! Quanto tempo è passato...

### 1.2 Come funziona PHP?

Una delle cose che più all’inizio genera disorientamento per chi non ha mai avuto a che fare con linguaggi di questo tipo è capire il concetto di *interpretazione*. PHP è un linguaggio di scripting interpretato server-side (*server-side HTML-embedded scripting language*). In parole povere, quando viene richiesta una

pagina web contenente codice PHP al suo interno, il web server (nel nostro caso faremo riferimento ad Apache) manda la pagina con il codice all'engine del PHP, il quale da come risposta delle richieste pagine HTML completamente formattate, rendendo il codice perfettamente trasparente all'utente finale. È un funzionamento molto semplice, efficace e veloce.

Visualizzando il codice delle pagine (l'estensione delle pagine PHP è .php, tranne diversi settaggi del web server) si vedrà solo HTML puro e di fatto il sorgente della pagina PHP può essere modificato solo dal web master (o da chi ha le credenziali per accedere al sorgente).

PHP attualmente viene eseguito come modulo all'interno del web server. Questo meccanismo, che permette sia maggiore sicurezza che maggiore efficienza, non è stato l'unico adottato. Le vecchie versioni di PHP venivano eseguite come script CGI. L'installazione dell'engine PHP è accuratamente spiegata, a seconda delle versioni e del tipo di web server, sul sito [www.php.net](http://www.php.net), nella sezione relativa alla documentazione. **La documentazione del PHP è scrupolosa e completa: deve essere sempre il vostro punto di riferimento!!!**

---

## 2. Concetti Fondamentali

### 2.1 PHP Escape

Il dubbio fondamentale: come fa PHP a capire cosa deve elaborare e cosa deve lasciare così com'è? La risposta è presto data: glielo si dice! :)

Senza fare troppi sofismi, vediamo subito un esempio:

```
<html>
  <head>
    <title>...</title>
  </head>

  <body>
    <!-- codice HTML non interpretato -->

    /* codice PHP interpretato */

    <!-- codice HTML non interpretato -->
  </body>
</html>
```

Listato 1: Pagina PHP base

Nella listato 1 si vede una riga che deve essere interpretata dall'engine PHP. Per indicarla si usano particolari sequenze di escape. La più diffusa è:

```
| <?php ... ?>
```

oppure:

```
| <? ... ?>
```

Esistono altre modalità, alcune delle quali obsolete. Consultate la documentazione se ne volete sapere di più!

## 2.2 Include & Require

Uno degli utilizzi più classici delle architetture Server Side è quello della creazione di pagine “dinamiche” costruite come mosaici, formate da tanti file che si legano tra loro formando il risultato finale. Essendo la risposta del server ad una richiesta di pagina .php puro codice HTML, questo sistema è un “trucco” molto elegante per modificare un certo numero di pagine con una sola modifica: è infatti necessario intervenire solo su un sottomodulo, invece che modificare un numero imprecisato di pagine, per ottenere il risultato cercato. HTML permette di ottenere un risultato simile tramite l'utilizzo dei frame, tecnica di web design ormai superata!!!

La sintassi per l'utilizzo di include e require è analoga:

```
| include('nomefile.ext');
```

**Domanda:** dov'è la differenza? Facile: se volete che lo script continui se manca un collegamento, usate `include`, altrimenti non esitate ad usare `require`.

Questo comportamento è dettato da un utilizzo importante di questi due costrutti: la realizzazione di file di libreria, contenenti una serie di funzioni, che vengono incluse ed utilizzate nel file principale di uno script. Così, se lo script può completare anche con la mancanza di una di queste librerie, si utilizza `include`, altrimenti è bene usare `require`.

Esistono altri due comandi importanti: `require_once` e `include_once`. Entrambi si comportano come già detto, ma in più controllano se un file è stato già incluso, eseguendo l'operazione una sola volta.

*Mio consiglio:* in genere la scelta migliore è usare `require_once`.

## 2.3 Operatori

Uno degli aspetti più importante, anche se molto semplice, di ogni linguaggio di programmazione è quello della corretta comprensione dell'uso e del significato degli operatori. Gli operatori sono quei caratteri particolari che indicano che tipo di operazione si deve svolgere. Gli operatori sono fondamentalmente di due tipi: aritmetici e logici. Queste due categorie però non sono le uniche!!!

Vediamo alcune tabelle con gli operatori più importanti, partendo dagli **operatori aritmetici**:

Operatore	Operazione eseguita
$\$a + \$b$	Somma tra $\$a$ e $\$b$
$\$a - \$b$	Resto di $\$b$ sottratto ad $\$a$
$\$a * \$b$	Prodotto di $\$a$ e $\$b$
$\$a / \$b$	Divisione tra $\$a$ e $\$b$
$\$a \% \$b$	Resto di $\$a$ diviso per $\$b$

Altra importante categoria è quella degli **operatori logici**:

Operatore	Operazione eseguita
<code>\$a AND \$b</code>	Vero se <code>\$a</code> e <code>\$b</code> sono veri entrambi
<code>\$a &amp;&amp; \$b</code>	Vero se <code>\$a</code> e <code>\$b</code> sono veri entrambi
<code>\$a OR \$b</code>	Vero se o <code>\$a</code> o <code>\$b</code> è vero
<code>\$a    \$b</code>	Vero se o <code>\$a</code> o <code>\$b</code> è vero
<code>\$a XOR \$b</code>	Vero se o <code>\$a</code> o <code>\$b</code> è vero, ma non entrambi
<code>!\$a</code>	Vero se <code>\$a</code> non è vero

Un'altra categoria di operatori fondamentale è quella degli **operatori di confronto**. Adesso vediamo i più importanti, ma quando si parlerà di variabili si farà riferimento a questa categoria spiegandone le caratteristiche più importanti.

Operatore	Operazione eseguita
<code>\$a == \$b</code>	Vero se il valore di <code>\$a</code> e <code>\$b</code> coincidono
<code>\$a != \$b</code>	Vero se il valore di <code>\$a</code> e <code>\$b</code> non coincidono
<code>\$a &lt; \$b</code>	Vero solo se <code>\$a</code> è più piccolo di <code>\$b</code>
<code>\$a &gt; \$b</code>	Vero solo se <code>\$a</code> è più grande di <code>\$b</code>
<code>\$a &lt;= \$b</code>	Vero se <code>\$a</code> è minore o uguale a <code>\$b</code>
<code>\$a &gt;= \$b</code>	Vero se <code>\$a</code> è maggiore o uguale a <code>\$b</code>

Ci sono due operatori particolari che meritano una trattazione a parte. Il primo è il **fondamentale** operatore di assegnamento. Vediamolo subito con un esempio.

```
| $a = 5;
```

L'operatore "=" ha la capacità di "copiare" il valore a destra del simbolo dentro la variabile a sinistra del simbolo (le due parti, in programmazione, vengono detti anche "valore destro" e "valore sinistro"). Questa proprietà rende possibile l'utilizzo delle variabili, a cui vengono assegnati dei valori o da cui vengono letti dei valori. Si tratta di una proprietà fondamentale in ogni linguaggio di tipo procedurale (ed il PHP è di fatto uno di questi!).

L'altro operatore che merita una trattazione a sè è l'operatore di concatenazione. Questo è particolarmente utile quando si lavora con le stringhe. Ad esempio:

```
| $tmp1 = "Rocker";
| $tmp2 = "Rock and Roll";
| $result = $tmp1 . " love " . $tmp2; // "Rocker love Rock and Roll"
```

In questo caso abbiamo concatenato una serie di stringhe (tramite l'operatore ".") per ottenere una stringa più lunga. Spesso lavorare in questo modo permette di creare le stringhe durante l'elaborazione, componendo la pagina a seconda dei diversi flussi d'elaborazione che l'esecuzione prende.

Come già detto, la quantità di operatori che il PHP possiede è grandissima. Rimando al manuale per una trattazione più completa e dettagliata. Come sempre, "che il manuale sia con te!".

## 3. Variabili

Uno degli aspetti base di ogni linguaggio di programmazione sta nelle variabili. Le variabili del PHP, in modo simile al Perl, ma diversamente dal C, sono molto flessibili e vengono riconosciute automaticamente. La variabile nel PHP viene definita nel suo contenuto dall'engine<sup>1</sup>. Le variabili possono contenere qualsiasi cosa. Imparerete con il tempo che la variabile del PHP può essere usata come un "notes" dove inserire una funzione che deve essere richiamata più volte.

L'inizializzazione di una variabile può essere fatta semplicemente assegnandole un valore (ricordate l'operatore =). Non deve essere definita a monte, ne tanto meno si deve definire di che tipo è. A volte però può essere utile e necessario farlo! Per assegnare anche il tipo ad una variabile si usa questa sintassi:

```
| $variabile = (int) 1;
```

Per sapere quali sono i tipi di variabili (da sostituire a posto di int) consultate il manuale del PHP. **Importante:** tutte le variabili del PHP iniziano con \$. Questa è una regola imprescindibile, ma anche una forma di notazione che permette subito di distinguere nel codice quello che cerchiamo.

Un array è un "insieme" di valori identificati da un solo "oggetto" (oddio, la definizione non è felicissima, ma passatemela!) L'inizializzazione di un array è fatta assegnandoli un valore... ma in modo diverso dalla pura assegnazione. Ecco un esempio:

```
| $nomi[ ] = "Davide" // Indicizzato come $nomi[0]  
| $nomi[ ] = "Fabio" // Indicizzato come $nomi[1]
```

...così via!

In questo modo quando viene inserito un nuovo valore nell'array, questo diventa l'ultimo. È importante ricordare che i membri di un array partono nel conteggio da 0 e non da 1! Altri linguaggi partono nel conteggio da 1, ma non - ad esempio - il C da cui il PHP è derivato.

Esistono array anche multidimensionali, differenti da quelli visti in precedenza che invece erano array monodimensionali. Ecco alcuni esempi di array multidimensionali:

```
| # Example 1:  
| $a["color"] = "red";  
| $a["taste"] = "sweet";  
| $a["shape"] = "round";  
| $a["name"] = "apple";  
| $a[3] = 4;  
  
| # Example 2:  
| $a = array(  
|     "color" => "red",  
|     "taste" => "sweet",  
|     "shape" => "round",  
|     "name" => "apple",  
|     3 => 4  
| );
```

---

<sup>1</sup>Chi ha una anche leggera infarinatura del C sa che invece lì le variabili devono essere tutte definite a monte del programma, in modo analogo a quanto avveniva con il Pascal. Nel C++ invece le variabili sono strettamente tipate, ma possono essere dichiarate ovunque nel codice.

Entrambi questi array costruiscono la medesima struttura, ma sono scritti in maniera differente. Se provassimo ad esempio a richiamare una funzione del genere:

```
| echo $a[name];
```

Quale sarebbe il risultato? A video comparirebbe:

```
apple
```

Ora vediamo un array multidimensionale più complesso:

```
| $a = array(  
  "apple" => array(  
    "color" => "red",  
    "taste" => "sweet",  
    "shape" => "round"  
  ),  
  "orange" => array(  
    "color" => "orange",  
    "taste" => "sweet",  
    "shape" => "round"  
  ),  
  "banana" => array(  
    "color" => "yellow",  
    "taste" => "paste-y",  
    "shape" => "banana-shaped"  
  )  
);
```

In questo caso si è creato un array multidimensionale più complesso. Per identificare un valore dell'array si devono richiamare 2 dimensioni... così:

```
| echo $a["apple"]["taste"];
```

E come output si avrebbe:

```
sweet
```