Web Scripting

PHP *(Hypertext Preprocessor)*

(Apache Web Server)

Tecnologie lato Client / Server

L'ambiente: Apache

Apache è il più popolare WebServer disponibile oggi.

La popolarità è dettata da alcune caratteristiche:

- Portabilità
- Scalabilità
- Costo

E' incluso, di ***default***, in numerose distribuzioni:

- Linux
- FreeBSD
- MacOS X
- Windows *(NT,2k,XP)*

Uno sguardo all'installazione

E' necessario configurare Apache

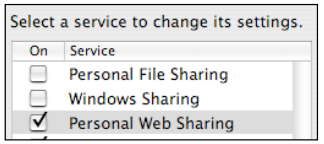
- La configurazione *può cambiare* tra piattaforme diverse
- Vedremo come configurarlo su un BSD *(MacOS X)*

E' semplice

- I files di configurazione seguono lo standard Unix

Avviare il Servizio

Apple Menu
System Preferences
Sharing



- Apache è configurato per utilizzare l'utente attuale
  - E' possibile modificare l'username da "*preferenze utente*"
- Se l'utente è "*valvoline*", la pagina personale sarà
  - http://127.0.0.1/~valvoline
  - http://localhost/~valvoline

Home Page e default settings

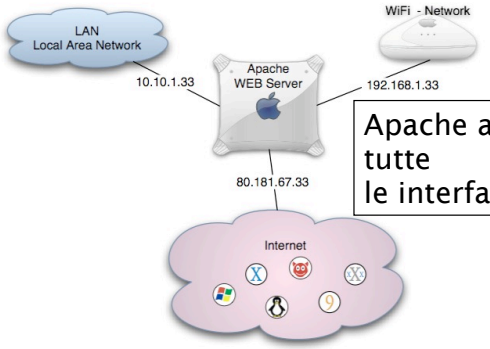
127.0.0.1 *o* **localhost** è un indirizzo speciale
- Ogni computer ne ha uno
- Rappresenta il computer stesso

All'indirizzo **127.0.0.1** *o* **localhost**, attraverso il Web-Browser
- risponde il server Apache locale

http://localhost/~valvoline
http://127.0.0.1/~valvolline

E' presente una pagina generica
- messa a disposizione dal server stesso
- verifica il corretto funzionamento del servizio

Interfaces Binding

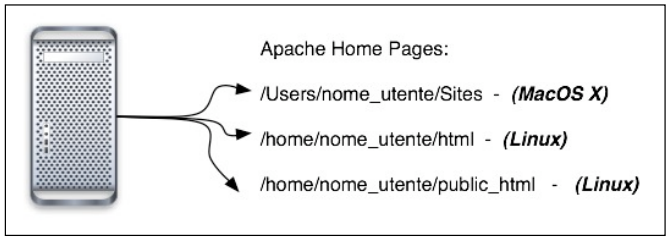


Apache ascolta le richieste su tutte
le interfaccie di rete disponibili.

Oltre all'indirizzo **127.0.0.1** saranno presenti tutti gli indirizzi configurati sul server.

Personal Web Page

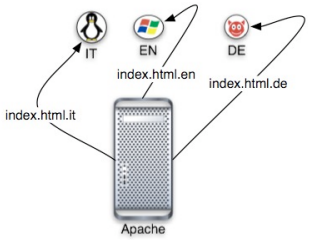
- Le pagine personali si trovano nel proprio spazio utente



- La pagina di default del server si trova all'indirizzo:
  - /Library/Webserver/Documents - (***MacOS X)***
  - /var/www - ***(Linux)***

Multi Language System

- E' possibile realizzare un sistema di pagine *multilinguaggio*



- Basterà aggiungere ogni pagina nella lingua desiderata
- Salvarla con estensione *nazionalizzata* (.de, .it, .en, etc.)
- Il sistema sceglierà la pagina adeguata in base al browser

La configurazione

Eseguendo, dal prompt, il comando: ***httpd -V***

Otterremo alcune informazioni riguardanti Apache

…
 -D SERVER_CONFIG_FILE="/etc/httpd/httpd.conf"
…

- Questo è il file di configurazione *generale* di Apache
  - Attraverso questo file si personalizza il WebServer

La configurazione: PHP

- PHP è molto popolare
- Ben supportato
- OpenSource
- Efficente e Veloce
- Installato di default con Apache*(spesso)*

- Il codice viene interpretato ed incluso nelle pagine *HTML*

- Comunemente, *Installare* PHP consiste
  - nell'includere alcune direttive
    - Presenti *(ma commentate)* nel file di configurazione

La configurazione: PHP (2)

 # LoadModule php4_module libexec/httpd/libphp4.so
 # AddModule mod_php4.c

- Abilitano / Disabilitano il caricamento del modulo *PHP*
  - Al prossimo riavvio del server Apache

- **Rimuovere** il cancelletto e **riavviare** il server

 # AddType application/x-httpd-php .php
 # AddType application/x-httpd-php-source .phps

- In sostanza, qualunque file con estensione .php
  - Deve essere processato dal modulo *PHP* del server

Il file di LOG

- Il file di log di Apache
  - Fornisce informazioni per ogni nuovo modulo aggiunto

[Tue May 11 16:05:04 2004] [notice] Apache/1.3.29 (Darwin) PHP/4.3.2 configured -- resuming normal operations
[Tue May 11 16:05:04 2004] [notice] Accept mutex: flock (Default: flock)

- Il modulo PHP è stato installato correttamente
- Apache ci dice che PHP è abilitato

Prova su strada

- Verifichiamo il corretto funzionamento di Apache e PHP:

Esempio:

```
<html><body>
    <h1>hello world!</h1>
    <?php
        phpinfo()
    ?>
</body></html>
```

- **phpinfo()** ritorna informazioni, circa l'ambiente e sue caratteris

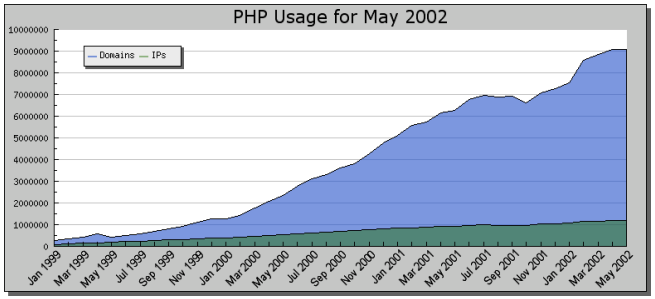
Software necessario

Ricapitolando, per utilizzare PHP serve:

- Un Web server
  - **Apache**
  - IIS *(Internet Information Server)*
  - Personal Web Server
- PHP4
- Un Browser
- Un Database *(opzionale)*
- Un po' di pazienza *(necessaria)*

Perché PHP ?

Perché PHP e non altro ?



ASP, JSP, RUBY, etc.

- Sono valide alternative, ma PHP le batte tutte!
- PHP si sposa egregiamente con Apache

Perché PHP ? (2)

Le ragioni per amare PHP:

- E' gratuito
- La combinazione PHP/Apache/MySQL gira su hardware vecchio.
- PHP, in prima battuta, fù rilasciato con licenza GPL *o "copyleft"*
- Adesso PHP4 ha la propria licenza *opensource*

Perché PHP ? (3)

Le ragioni per amare PHP:

- E' facile
  - Non sono richieste profonde conoscenze del SO
  - Perl è un linguaggio "di sola scrittura"
  - Molte delle funzioni sono predefiniti
  - E' un linguaggio OOB
- E' embedded
  - Le pagine PHP sono comuni pagine HTML
  - Si "sconfina" nella modalità PHP solo quando necessario

Perché PHP ? (4)

PHP è multipiattaforma

- Gira su sistemi UNIX e Windows
- E' compatibile con I maggiori Server Web Mondiali:
  - Apache
  - IIS *(Microsoft Corporation)*
  - iPlanet Server *(Netscape Enterprise)*

PHP è stabile

- Il server non deve essere riavviato spesso
- Non cambia radicalmente tra versioni diverse

PHP è veloce

- E' molto più veloce rispetto i migliori scripts CGI
- Gli scripts *C-Based* devono creare un processo ogni volta

Un esempio più interessante

- Un esempio più interessante, potrebbe essere il seguente:

Esempio:

```
<html><body>
    <h3>your browser is: </h3>
    <?php
        echo $HTTP_USER_AGENT;
    ?>
</body></html>
```

Output:

your browser is: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/124 (KHTML, like Gecko) Safari/125.1

Perché Embedded?

HTML Embedding:

```
<HTML><HEAD>
<TITLE>Search results for "<?php print $query; ?>"</TITLE>
</HEAD><BODY>
```

Programmazione Tradizionale (CGI):

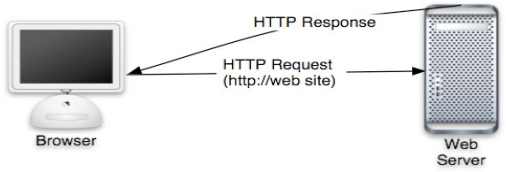
```
#!/usr/bin/perl

print "<HTML><HEAD>\n";
print "<TITLE>Search results for \"$query\"</TITLE>\n";
print "</HEAD>\n";
print "<BODY>\n";
```

Cosa è l'HTML Dinamico?

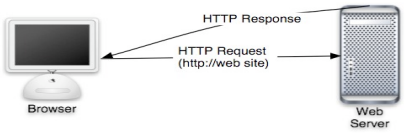
- Distinzione fondamentale tra pagine "statiche" e "dinamiche"
- Dinamico può significare tutto ciò che non è HTML puro!
- Descrive funzioni lato client e funzioni lato server
  - Sul Client
    - Titoli scorrevoli
    - Pagine che si autoaggiornano
    - Elementi che scompaiono e riappaiono
  - Sul Server
    - Denota contenuti assemblati in tempo reale

HTML: Static vs Dynamic

- Il tipo di pagina Web più elementare è completamente **statico**
  - http://www.dmi.unict.it/~pistagna
- Il client ***effettua*** una richiesta al server
- Il server ***risponde*** con la pagina HTML *(via TCP/IP)*



HTML: Static vs Dynamic (2)



Alcuni vantaggi:

- Qualsiasi Browser può visualizzarlo correttamente
- I dispositivi embedded possono visualizzarlo correttamente
- Le richieste sono soddisfatte velocemente
- HTML è facile da imparare

Tecnologie lato client

Per sopperire alle mancanze dovute alla staticità

- Le aggiunte più comuni sono sul ***lato client***
  - Cascading Style Sheets ***(CSS)***
  - JavaScripting Lato Client
  - Applets Java
- Il supporto per queste tecnologie è
  - All'interno del Browser

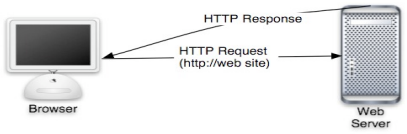
Tecnologie lato client (2)

L'HTML Statico e le tecnologie lato client, hanno alcuni svantaggi:

- Scalabilità limitata
- Interattività con il server difficile
- Inclusione di Metadati molto complessa
- Non tutti I clients vedranno lo stesso risultato
- Ci sono casi in cui il loro uso è indispensabile
  (RealTime 3D, Gfx, Multimedia)

Tecnologie lato server

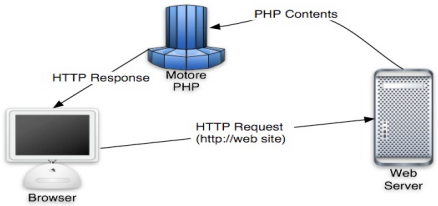
Lo schema precedente resta, in generale, valido



La novità si presenta all'atto di preparare una pagina

- Il server si imbatte in uno script (script lato server)
- Lo script viene identificato dal server scorrendo la pagina
  - Vengono ricercati degli identificatori: <?php, ?>
- Il contenuto viene elaborato dal server e rispedito al browser

Tecnologie lato server (2)




```
<HTML><BODY>
Numero di giorni trascorsi: <?php echo gmDate("z"); ?>
</BODY></HTML>
```

```
<HTML><BODY>
Numero di giorni trascorsi: 158
</BODY></HTML>
```


Tecnologie lato server: La Cache

Esiste un potenziale problema nella scrittura di codice PHP:

- **La Cache**
  - Memorizza le pagine web già visionate
  - Limita *l'interattività* dell'utente e dei contenuti dinamici
  - Non utilizzerete in nessun modo il codice dinamico
  - L'informazione restituita risulterà ***non*** corretta
  - A volte non basta neanche premere il pulsante ***Refresh***

Tecnologie lato server: La Cache (2)

Per risolvere il problema esiste un insieme di intestazioni HTTP:

```
<?
header("Cache-Control: no-cache, must-revalidate");
header("Pragma: no-cache");
?>
```

Aggiunto all'inizio di un documento PHP, evita il fastidioso problema del ***Refresh*** della pagina.

Il PHP nelle pagine HTML

- PHP, quindi, è un linguaggio **embedded**
  - Viene introdotto nella pagine HTML con appositi TAG.
- Una porzione di codice PHP inizia con: **<?php** *o* **<?**
- Termina con: **?>**

Esempio:

```
<?php phpinfo(); ?>
```


Il PHP nelle pagine HTML (2)

- Tutto il testo esterno ai simboli **<?php** e **?>** è considerato *HTML*.
  - Viene spedito al client senza modifiche.
- Il testo contenuto all'interno dei simboli **<?php** e **?>**
  - E' codice *PHP* e viene gestito dal PreProcessore *PHP*.
- Le pagine vengono interpretate, elaborate, modificate
  - Soltanto successivamente vengono inviate al Browser

Il PHP nelle pagine HTML (3)

- Il codice PHP viene eseguito sul server
  - *Non* viene inviato al client
- Non è possibile visualizzare il codice PHP sul client.
  - E' una *prima forma di sicurezza* dell'applicativo.
- Il codice PHP può effettuare output di testo.
  - L'output viene inviato al client
  - Al posto del codice PHP.
- L'output *deve* essere in formato HTML.

Il PHP nelle pagine HTML (4)

```
<I>ABC</I>
<?php
echo "<BR>"
?>
testo
```


```
<I>ABC</I>
<BR>
testo
```


Il PHP nelle pagine HTML (5)

Esempio:

```
<HTML><BODY>
Oggi &egrave; il giorno
<?php echo gmdate("M d Y"); ?>
</BODY><HTML>
```

Viene ricevuto come:

```
<HTML><BODY>
Oggi &egrave; il giorno
Mar 25 2002
</BODY><HTML>
```


Istruzioni e Blocchi

- Ogni istruzione PHP e' delimitata dal simbolo "**;**"
- Su ogni una riga vi possono essere più istruzioni
  - Ciascuna istruzione può estendersi su più righe.
- Le istruzioni possono essere raggruppate in blocchi
  - I blocchi sono delimitati dalle parentesi graffe " **{ }** ".

Istruzioni e Blocchi (2)

Esempio:

```
<HTML><BODY>
<?php
if (($User == "php") && ($Passwd == "secret"))
{ echo "<H1> Logging in</H1>";
}
else
{ echo "<H1>Access denied";
  echo "<BR> </H1>";
}
?>
</BODY><HTML>
```

Sintassi C-Like

```
<?
    //semplice ciclo for
    for ($loop = -5; $loop < 5; $loop++) {
        echo "$loop<BR>\n";
    }
    //cicliamo sull'accumulatore $loop
    while(--$loop) {
        //in un sol colpo otteniamo valori 0/1
        switch(++$i%2) {
            case 0: echo "Even<BR>\n";break;
            case 1: echo "Odd<BR>\n";break;
        }
    }
    //ultimo tipo di ciclo con post-controllo
    do {
        echo "$loop<BR>\n";
    }while (++$loop < 10);
?>
```


Un altro esempio (interessante)

```
<?php
     $google =
     preg_quote("http://www.google.it", '/');
     if (preg_match("/^$google.*/i",
          $_SERVER['HTTP_REFERER']))
               print "ciao google";
?>
```

- Controlliamo il ***referer*** che ci ha linkato
- Potremmo svolgere azioni diverse a seconda di chi ci linka
- Questo è solo un'esempio di quanto è utile/potente la **programmazione dinamica**

Commenti

I commenti monoriga utilizzano due tipi di marcatori:

Esempio:

```
<?php
// echo "<H1>riga commentata";
# echo "<H1>altra riga commentata";
echo "<P>riga non commentata</H1>";
?>
```


Commenti (2)

Commenti a blocco:

Esempio:

```
<?php
/*
echo "<H1>righe commentate";
echo "<P>righe commentate</H1>";
*/
?>
```

Variabili

- In PHP le variabili sono lo strumento utilizzato per memorizzare i dati all'interno dei programmi.
- I nomi delle variabili in PHP seguono le seguenti regole:
  - Le variabili sono sempre precedute dal simbolo "**$**" .
  - I nomi delle variabili possono essere costituiti con i caratteri alfabetici (minuscoli e maiuscoli), cifre e "**_**" .

Esempio:

```
<?php $nome="Ivan";  echo $nome; ?>
```


Variabili (2)

- Le variabili ***non*** possono incominciare per numero (solo per lettera o "_").
- Le variabili ***sono*** case-sensitive: lettere maiuscole o minuscole fanno differenza!
- Le variabili vengono assegnate utilizzando il simbolo "**=**".

Esempio:

```
$nome = "Lisa";
$prezzo = 21.95;
```


Variabili (3)

- Il valore di una variabile è il valore della sua assegnazione più recente.
- Le variabili ***non*** hanno bisogno di essere dichiarate prima dell'assegnazione.
- Le variabili non hanno tipo oltre a quello del valore corrente.
- Le variabili utilizzate prima di essere assegnate hanno valori di ***default***

Esempio:

```
$prezzo = 21.95; // questo è un float?
```


Variabili (4)

- Se si vuole essere avvertiti di variabili che non sono state assegnate, si deve modificare il *livello di segnalazione di errore*:

```
<?
error_reporting(15);
?>
```

- Si può anche modificare il file ***php.ini*** per impostare il livello di segnalazione di errore predefinito.

Variabili (5)

I valori contenuti nelle variabili vengono richiamati semplicemente indicando il nome della variabile stessa.

Esempio:

```
$Netto = 25;
$Tara = 7;
$Lordo = $Netto + $Tara;
echo $Lordo;
```


Variabili (6)

- Le variabili vengono definite automaticamente al loro primo utilizzo.
- A differenza di altri linguaggi, non è prevista la dichiarazione esplicita del tipo prima del loro utilizzo.
- Le variabili ***sono sempre locali***
- Esistono costrutti particolari del linguaggio per dichiarare ***variabili globali***.

Variabili (7)

Le variabili possono essere ridefinite. In tal caso può cambiare il loro tipo.

Esempio:

```
$citta = "Catania";
$citta = 95100;
```


Variabili (8)

- Le variabili possono essere usate prima di essere definite.
  - Sono di tipo unknown
  - Vengono trattate come variabili col valore 0 o stringhe vuote.

Esempio:

```
echo $citta;
$citta = 95100;
```

genera: **NULL**

Variabili: IsSet

- Le variabili non devono essere dichiarate prima dell'utilizzo.
- PHP fornisce una funzione che esamina una variabile per vedere se le è stato assegnato un valore: **IsSet()**

```
<?php $set_var = 0; // è stato assegnato un valore
if(IsSet($set_var)) print("set_var è impostata.<BR>");
else print("set_var non è impostata.<BR>");
if(IsSet($never_var)) print("never_var è impostata.<BR>");
else print("never_var non è impostata.<BR>");
?>
```


Variabili: unset

- L'atto di distruggere una variabile ***è irrevocabile***
- La funzione **unset()** riporta una variabile ad uno stato di ***non assegnamento***, senza badare all'assegnazione precedente

```
<?php $set_var = 0; // è stato assegnato un valore
unset($set_var); // revochiamo tutto!
if(IsSet($set_var)) print("set_var è impostata.<BR>");
else print("set_var non è impostata.<BR>");
?>
```


Variabili: empty

- E' l'opposto logico di **IsSet()**
- Restituisce:
  - 1 se non c'è alcuna variabile (*o* == **0**, *o* == **" "**)
  - non indica nulla se la variabile esiste.

```
<?php $set_var = 0; // è stato assegnato un valore
echo empty($set_var);
unset($set_var);
echo empty($set_var);
?>
```


Variabili d'ambiente

- Sono variabili esistenti al di fuori di script PHP
- Sono disponibili in ***qualsiasi*** script PHP
- Forniscono informazioni sulla transazione Client/ Server
- Hanno lo stesso formato delle *normali* variabili

Variabili d'ambiente

- Non richiedono intervento da parte dell'utente
- Si possono vedere usando la funzione **`phpinfo()`**
- E' possibile fare riferimento alle *singole variabili*:
  - echo **$HTTP_COOKIE_DATA;** //contenuto dei cookies
  - echo **$HTTP_USER_AGENT;** //nome browser

Variabili d'ambiente (2)

- Ci sono molte altre variabili d'ambiente utili:
  - **$HTTP_FROM;** // contiene l'indirizzo email dell'utente che effettua una richiesta
  - **$HTTP_ACCEPT;** // contiene un elenco dei vari tipi di media-contents che l'utente è in grado di gestire
- E' possibile modificare queste variabili
  - non sono costanti
  - in generale, però, servono come riferimento

Ambito delle Variabili

- Qualsiasi variabile che non si trova all'interno di una funzione ha ambito ***globale***.
- Se si assegna una variabile all'inizio di un file PHP, il nome della variabile ha lo stesso significato per tutto il resto del file.
- L'assegnazione di una variabile ***non influirà*** sul valore delle variabili con lo stesso nome in altri file PHP.
- Esistono metodi per passare variabili tra un file e l'altro
  - utilizzando ***GET*** e ***POST***
  - utilizzando i ***cookies***
  - utilizzando un database

Ambito delle Variabili (2)

```
<HTML><HEAD>
<?php
$username="lisa"; // ambito globale
?>
</HEAD>
<BODY>
<?php
print("$username<BR>"); // l'ambito è sempre lo stesso
?>
</BODY></HTML>
```

Ambito delle Variabili (3)

E' una regola generale di PHP

- L'unico effetto dei tag è quello di permettere al motore di php di sapere
  - Se si vuole che il codice venga interpretato come PHP o che passi per intero come HTML.
- Sentitevi liberi di utilizzare i tag per passare da un modo all'altro secondo ***la convenienza***
- Passando da un modo all'altro non si perde in prestazioni!

Tipi di Dati

I dati contenuti nelle variabili sono tipati.

In particolare, i tipi che PHP può trattare sono:

- string
- integer
- double
- array
- object
- unknown

Tipi di Dati (2)

- I tipi dei dati vengono inferiti dal contesto. ***(dynamic typing)***
- Non è necessario dichiarare il tipo di una variabile.
  - Prima regola: ***non preoccuparsi***.
- Esistono comunque delle ***operazioni di casting*** con cui è possibile convertire un tipo di dato in un altro.
- In alcuni casi le conversioni sono ***automatiche***.

Tipi di Dati (3)

Esempio:

```
$nome = "Lisa";
$prezzo = 21.95;

echo $nome;
echo $prezzo;
echo $tmp;
```

Conversione di Tipi Automatica

- PHP fa un buon lavoro di conversione automatico
  - Farà la *cosa giusta*, eseguendo calcoli con tipi diversi

Esempio:

```
$asse = 8 + 4.523
```

- Sarà un ***float***
- L'intero **8** viene convertito, implicitamente, in **8.00** prima dell'operazione.

Tipi Assegnati per Contesto

Esempio:

```
$asse = substr(12345, 2, 2);
print("asse e' $sub<BR>");
```

- ***substr*** prende una stringa e restituisce una sottostringa.
  - Punto iniziale e lunghezza sono i due parametri successivi.
- Nessun Errore se passiamo un intero: **12345**
- PHP **converte** l'intero in stringa e poi **effettua** l'operazione!

I Numeri

- I tipi di dato ***integer*** contengono numeri interi.
- I tipi di dato ***double*** contengono numeri reali a doppia precisione (numeri in virgola mobile).
- Numeri privi di virgola vengono automaticamente considerati ***integer***.

Esempio:

```
$Quantita = 15;
```


I Numeri (2)

- Quando nei valori è presente la virgola, i numeri vengono considerati come ***double***.
- Stessa cosa per le frazioni.

Esempio:

```
$Costo = 6.73;
$Costo = 6/5;
```

I Numeri (3)

- Gli interi possono essere letti in tre formati:
  - Decimale (base 10) - *predefinito*
  - Ottale (base 8) - *specificato con uno **'0'** iniziale*
  - Esadecimale (base 16) - *specificato con un **'0x'** iniziale*

- Il valore di lettura ha effetto solo sulla conversione ***in fase di lettura!***
- All'interno i numeri sono conservati in ***formato binario***.

I Numeri (4)

Esempio:

```
<?
   $int_10 = 1000;
   $int_8 = 01000;
   $int_16 = 0x1000;
   print("int_10: $int_10<br>");
   print("int_16: $int_16<br>");
   print("int_8: $int_8<br>");
?>
```

Output:

```
int_10 = 1000;
int_16 = 4096;
int_8 = 512;
```


I Numeri (5)

I **double** sono numeri a virgola mobile:

Esempio:

```
$float1 = 345.67;
$float2 = 987,457;
$float3 = 3.0;
```

Attenzione!

Il fatto che ***$float3*** sia un numero "tondo" non lo rende un intero!

I Numeri (5)

Bisogna stare attenti a non confondere

- Il punto decimale con
  - L'operatore di concatenazione di stringhe.

Esempio:

```
echo 2 . 2;  // Ci sono due stringhe "2"
echo 2.2;    // La stringa è "2.2"
```

I Booleani

- Sono valori **TRUE** o **FALSE**
- Vengono utilizzati in strutture di controllo
- Possono essere combinati usando operatori logici
- E' una nuova aggiunta di **PHP4**
  - **PHP3** non aveva tipi booleani distinti
  - trattava certi valori di altri tipi come veri o falsi

```
<?php
if(TRUE) print("questo lo stampiamo<BR>");
else print("questo, invece, NO!<BR>");
?>
```


I Booleani (2)

- E' possibile determinare il ***vero*** di ogni valore che non è già un booleano
  - Se il valore è un **numero**: è falso se è 0; vero in tutti gli altri casi.
  - Se il valore è una **stringa**: è falso se la stringa è vuota o la stringa è "0"; vero in tutti gli altri casi.
  - Se il valore è un **tipo composto** (array): è falso se non contiene altri valori; è vero in tutti gli altri casi.

I Booleani (3)

```
<?php
      if(!$floatval) { $floatval = 1; }
      else $floatval++;
?>
```

```
<?php
      if(!$mystring) { $mystring = "Hello, World!"; }
      else $mystring.="-this chunk was added";
?>
```


I Booleani (4)

```
<?php
$floatval = sqrt(2.0) * sqrt(2.0) - 2.0;
if($floatval)
print("i float come booleani sono pericolosi!<BR>");
else print("per questa volta ha funzionato!");
?>
```

- La regola 1, dice che il float 0,0 deve essere convertito in un booleano falso.
- E' pericoloso utilizzare espressioni a virgola mobile
- Possibili errori di arrotondamento!

Le Stringhe

- Le stringhe sono il tipo di dato ***più importante*** in PHP.
- Sono sequenze di caratteri
- Le stringhe possono contenere intere frasi e vengono definite utilizzando virgolette **doppie** o **singole**.

Esempio:

```
$citta = "Catania";
$username = "Lisa";
```


Le Stringhe (2)

- Nelle stringhe possono essere inserite variabili.
- Se si inserisce un nome di variabile in una stringa
  - viene sostituito con il suo valore

Esempio:

```
$Nome = "Lisa";
$Saluto = "Ciao $Nome !";
// $saluto = "Ciao " . $Nome . "!";
echo $Saluto;
```

genera: **Ciao Lisa**

Le Stringhe (3)

- Stringhe tra virgolette **semplici** vengono trattate *quasi* letteralmente.
- Stringhe tra virgolette **doppie** sostituiscono variabili con i valori associati.
  - interpretano sequenze particolari di caratteri.

Esempio:

```
$Nome = "Lisa";
$Saluto ='Ciao $Nome !';
echo $Saluto;
```

genera: **Ciao $Nome**

Le Stringhe (4)

- Le stringhe possono essere concatenate utilizzando il simbolo punto "**.**"

Esempio:

```
$Nome = "Mario";
$Cognome = "Rossi";
$Nome_completo = $Cognome . " " . $Nome;
echo $Nome_completo;
```

genera: **Rossi Mario**

Regole di Concatenamento

- Se compare un "$" in una stringa tra doppi apici
- PHP tenta di interpretare il contenuto come variabile
  - Se la variabile è impostata su un **valore stringa**, la stringa viene concatenata
  - Se la variabile è impostata su un **valore non stringa**, il valore viene convertito in una stringa e quindi concatenato
  - Se la variabile **non è impostata**, PHP non concatena niente

Casting

- È possibile convertire un numero in stringa e viceversa utilizzando una operazione di ***cast***.
- Un cast si effettua scrivendo tra ***parentesi tonde*** prima della variabile, il tipo in cui si vuole trasformarla:

Esempio:

```
$Anno = "Anno " . (string)$year;
$Quantita = (integer)$Qty;
```

- Raramente il casting è necessario. Nella conversione verso numeri può produrre *risultati errati*.

Le Costanti

- Oltre alle variabili, PHP permette anche di definire delle ***costanti***.
- I nomi delle costanti sono generalmente scritti in maiuscolo e ***non cominciano*** con il simbolo **"$"** .
- Il valore delle costanti può essere ***soltanto letto***.

Le Costanti (2)

- Le costanti si assegnano utilizzando il comando **`define()`**.
- Il valore di una costante non può essere modificato.

Esempio:

```
define("NATALE", "25 dicembre");
echo NATALE;
```

Il comando `echo`

- Il comando **`echo`** è il comando principale con cui si scrive la pagina HTML generata dallo script PHP.
- Esso aggiunge una stringa alla pagina HTML restituita al client.
- Nelle pagine HTML ricevute dal client le righe scritte da **`echo`** vengono inserite al posto del codice php.
- E' leggermente più efficiente di **`print()`**

Il comando `echo`

- A differenza degli altri comandi, **`echo`** non richiede parentesi per specificare i parametri.
- Aggiunge alla pagina la stringa specificata subito dopo la parola chiave.

Esempio:

```
echo "<HTML><BODY><H1>Titolo dell'opera"
echo "</H1></BODY></HTML>";
```


Il comando `print`

- Anche il comando **`print()`** stampa una stringa.
- **`print()`** accetta **un solo** argomento e restituisce un valore, per informare se l'operazione di output ha avuto successo.
- E' possibile gestire l'output di print in maniera personalizzata.
- Ci torneremo più avanti nel corso.

Gli Operatori

Gli operatori aritmetici sono gli stessi del C:

- somma +
- sottrazione –
- divisione /
- moltiplicazione *
- modulo %

Le parentesi modificano l'ordine di precedenza degli operatori *(come in C)*.

Gli Operatori (2)

$numero = $numero + 1;

è equivalente a

++$numero;

Come in C esistono anche:

++$var, $var++

--$var, $var--

Gli Operatori (3)

$numero = 1;

Versione prefissa (incremento, poi assegnamento):

$risultato = ++$numero;

$risultato ⇒ 2

$numero ⇒ 2

Versione postfissa (assegnamento, poi incremento):

$risultato = $numero++;

$risultato ⇒ 1

$numero ⇒ 2

Ottimizzare il codice

- Utilizzare variabili temporanee è una **pessima** abitudine.
- L'uso di variabili temporanee **rallenta** l'esecuzione.
- Il tempo può incrementare del ***20%*** / ***25%***
- La presenza di troppe variabili temporanee può **ridurre la visibilità** del codice.

Esempio:

```
<?
  $tmp = date('r');
  echo $tmp;
?>
```

Esempio:

```
<?
   echo date('r');
?>
```


Ottimizzare il codice (2)

- Se il valore contenuto / utilizzato è richiesto più volte
  - L'uso delle variabili temporanee diventa **obbligo**!

Esempio:

```
<?
  for($i=0; $i<count($myarray); $i++)
      echo $myarray[$i];
?>
```

Esempio:

```
<?
  for($i=0, $tmp=count($myarray); $i<$tmp; $i++)
      echo $myarray[$i];
?>
```

Ottimizzare il codice (3)

Esempio:

```
<?
   $i = 5;
   $j = &$i;

   echo “i: “ . $I . “\n”;
   echo “j: “ . $j . “\n”;
?>
```

- **$j** è un **alias** per **$i**.
- PHP4 vede la stessa variabile *(passaggio per referenza)*
- Cambiando valore a **$i** o **$j**, l’altra avrà lo stesso valore.
  - Puntano alla stessa zona di memoria.
- I puntatori ***esistono*** anche in PHP!!

Ottimizzare il codice (4)

- In PHP4, esiste il ***Reference Counting***
- Elimina allocazione di nuova memoria inutile.

```
<?
   $i = 5;  //RC[5]=1
   $j = $i; //RC[5]=2
?>
```

1

```
<?
   $i = 5;  //RC[5]=1
   $j = $i; //RC[5]=2
   $j = 7;  //RC[5]=1 && RC[7]=1
?>
```

2

```
<?
   $j = 7;  //RC[5]=1 && RC[7]=1
   $i = $j; //RC[5]=0 && RC[7]=2
?>
```

3

Ottimizzare il codice (5)

- Fino a quando non si cambia il valore della variabile
  - Il passaggio per valore permette di sfruttare l’RC
    - **performance++**
- Per decidere se usare passaggio per valore o riferimento:
  - Gli oggetti dovrebbero essere passati per riferimento.
  - Ogni altra cosa, **inclusi gli array**, per valore.

Operazioni sulle Stringhe: chop()

```
Synopsis:
      chop(string)
```

Elimina gli spazi vuoti **finali** dalla stringa indicata.

Valore di ritorno: Stringa

```
<?
   echo chop("vorrei togliere gli spazi finali    ");
?>
```

Operazioni sulle Stringhe: chunk_split()

```
Synopsis:
     chunk_split(string [,length] [,end])
```

Suddivide la stringa in porzioni di *length* caratteri.

Inserendo la stringa *end* come delimitatore tra ogni porzione.

```
<?
   echo chunk_split("pinopenapanepone", 4, "#\n");
?>
```


Operazioni sulle Stringhe: ltrim()

```
Synopsis:
     ltrim(string)
```

Elimina gli spazi vuoti all'inizio della stringa specificata.

Valore di ritorno: **string**

```
<?
   echo ltrim("   Hello, World!");
?>
```


Operazioni sulle Stringhe: rtrim()

```
Synopsis:
     rtrim(string)
```

Elimina gli spazi vuoti alla fine della stringa specificata.

Valore di ritorno: **string**

```
<?
   echo ltrim("Hello, World!     ");
?>
```


Operazioni sulle Stringhe: trim()

```
Synopsis:
     trim(string)
```

Elimina gli spazi vuoti all'inizio ed alla fine della stringa specificata.

Valore di ritorno: **string**

```
<?
   echo ltrim("      Hello, World!     ");
?>
```

Operazioni sulle Stringhe: ucfirst()

```
Synopsis:
     ucfirst(string)
```

Converte in maiuscolo il primo carattere di **string**.

Valore di ritorno: **string**

```
<?
   echo ucfirst(“hello, world!”);
?>
```


Operazioni sulle Stringhe: ucwords()

```
Synopsis:
     ucwords(string)
```

Converte in maiuscolo il primo carattere di ogni parola di **string**.

Valore di ritorno: **string**

```
<?
   echo ucwords(“hello, world!”);
?>
```


Operazioni sulle Stringhe: substr_replace()

```
Synopsis:
   substr_replace(string, repl, start [,len])
```

Sostituisce con **repl** il testo all’interno di **string** partendo da **start**, per **len** caratteri.

```
<?
   echo substr_replace("Hello!,World!","GoodBye,",0,8);
?>
```


Operazioni sulle Stringhe: substr()

```
Synopsis:
   substr(string, start [,len])
```

Restituisce **len** caratteri da **string**, partendo dalla posizione **start**.

```
<?
   echo substr("Hello!,  World!", 0, 6);
?>
```

Operazioni sulle Stringhe: strtoupper()

```
Synopsis:
   strtoupper(string)
```

Converte la stringa specificata in caratteri maiuscoli.

Valore di ritorno: **string**

```
<?
   echo strtoupper("Hello, World!");
?>
```


Operazioni sulle Stringhe: strtolower()

```
Synopsis:
   strtolower(string)
```

Converte la stringa specificata in caratteri minuscoli.

Valore di ritorno: **string**

```
<?
   echo strtolower("Hello, World!");
?>
```


Operazioni sulle Stringhe: strlen()

```
Synopsis:
   strlen(string)
```

Restituisce la lunghezza della stringa specificata.

Valore di ritorno: **integer**

```
<?
   echo strlen("Hello, World!");
?>
```


Leggere i campi dai forms

- Non è presente un meccanismo di ***prompt***
  - PHP deve utilizzare i campi di un ***form HTML*** per l'input.
- In PHP *(< 4.2.x)* è estremamente facile accedere al contenuto dei campi di un form:
  - Ad ogni campo corrisponde una ***variabile visibile*** all'interno del codice PHP.
- Le nuove versioni di PHP, **scoraggiano** quest'utilizzo.
  - Al posto di una variabile per ogni campo, viene creato un array di valori, ***accessibili per nome campo***.

Leggere i campi dai forms (2)

- Quando l'utente preme ***submit*** in un ***form***
- Il server esegue lo script/il file indicato dall'attributo *ACTION*

Esempio:

```
<FORM ACTION="Search.php" METHOD="GET" NAME="Ricerca">
      <INPUT TYPE="TEXT" NAME="Parole">
      <INPUT TYPE="SUBMIT">
</FORM>
```


Leggere i campi dai forms (3)

- Il parametro **METHOD** impartisce istruzioni sul **come** passare i parametri.

Esempio:

```
<FORM ACTION="Search.php" METHOD="GET" NAME="Ricerca">
      <INPUT TYPE="TEXT" NAME="Parole">
      <INPUT TYPE="SUBMIT">
</FORM>
```


Leggere i campi dai forms (4)

- Lo script PHP *(che viene chiamato)* ritroverà in opportune variabili i valori immessi nei campi dall'utente.
- Esistono due modi con cui possono essere inviati i dati:
  - Il metodo **GET**
  - Il metodo **POST**
- È possibile passare i parametri ad uno script php attaccando all'indirizzo dello script le coppie variabile/ valore.
- Altri metodi di passaggio parametri:
  - **Cookies**
  - **Sessioni**

Leggere i campi dai forms: GET

- Il metodo **GET** passa argomenti da una pagina alla successiva come parte della stringa ***(URI)***
- Allega i nomi e i valori delle variabili
- Un punto di domanda **'?'** come separatore dell'URL
- Una **'&'** commerciale come separatore di argomenti
- Coppia argomento / valore, separata da **'='**

Esempio:

http://www.miosito.org/index.php?**indice=1&parametro=5**

Leggere i campi dai forms: GET (4)

- **GET** non è adatto per i login.
- Nome utente e password viaggiano visibilmente sull'URL.
- Ogni invio **GET** viene registrato nel file di registro del server WEB, compresi i dati.
- La lunghezza dell'URL **è limitata**.
- E' necessario per la gestione, ad esempio, dei **Bookmarks!**

Esempio:

http://www.miosito.org/index.php?**login=mrossi&pass=mrossi**

http://www.miosito.org/index.php?**section=qualcosa&sito=altro**

Leggere i campi dai forms: POST

- E' il metodo **preferito** di gestione dei moduli.
- Il set dei dati è ***incluso*** nel corpo del modulo quando viene inviato.
- Non c'e' ***cambiamento*** visibile nell'URL.

Esempio:

http://miosito.org/index.php - ***(POST)***

http://miosito.org/index.php?login=mrossi&pass=mrossi - ***(GET)***

Leggere i campi dai forms: POST (2)

- E' più sicuro di GET!
  - Le informazioni di query ***non sono*** mai parte dell'URL
- C'è un limite ***molto più ampio*** nella quantità di dati che possono essere passati.
- I risultati non possono essere inseriti nel **Bookmark**.

Esempio:

http://miosito.org/index.php - ***(POST)***

http://miosito.org/index.php?login=mrossi&pass=mrossi - ***(GET)***

Leggere i campi dai forms (5)

- Il valore di un campo viene inserito in una variabile avente lo stesso nome ***(deprecated)***.
- Viene inserito in un array di parametri, accessibile

mediante:

- **$_REQUEST**["nomevar"] - *(per post e get)*
- **$_GET**["nomevar"] - *(per get)*
- **$_POST**["nomevar"] - *(per post)*

Leggere i campi dai forms (6)

- Dobbiamo ***cambiare*** il modo di ***accedere*** alle variabili:

Esempio:

```
<FORM ACTION="Search.php" METHOD="GET" NAME="Ricerca">
      <INPUT TYPE="TEXT" NAME="Parole">
      <INPUT TYPE="SUBMIT">
</FORM>
```

Search.php:

```
<HTML><BODY><H1>
      Stai cercando: <?php echo $Parole; ?>
</HTML></BODY></H1>
```


Leggere i campi dai forms (7)

- **$_REQUEST** indicizza tutti i parametri.

Esempio:

```
<FORM ACTION="Search.php" METHOD="GET" NAME="Ricerca">
      <INPUT TYPE="TEXT" NAME="Parole">
      <INPUT TYPE="SUBMIT">
</FORM>
```

Search.php:

```
<HTML><BODY><H1>
      Stai cercando:
      <? echo $_REQUEST["Parole"]; ?>
</HTML></BODY></H1>
```


Leggere i campi dai forms (8)

- se usiamo **GET**, c'è un array corrispondente!

Esempio:

```
<FORM ACTION="Search.php" METHOD="GET" NAME="Ricerca">
      <INPUT TYPE="TEXT" NAME="Parole">
      <INPUT TYPE="SUBMIT">
</FORM>
```

Search.php:

```
<HTML><BODY><H1>
      Stai cercando:
      <? echo $_GET["Parole"]; ?>
</HTML></BODY></H1>
```


Leggere i campi dai forms (10)

- I campi **CHECKBOX**
  - Prendono il valore dell'attributo **VALUE**
    - *Se l'opzione è stata selezionata.*
  - Se **VALUE** è omesso, il valore di default è "**on**". *(string)*
- Se la casella non è stata selezionata.
  - La variabile contiene una stringa vuota **" "**.

Leggere i campi dai forms (11)

- Nei campi **RADIO**
  - Viene inserito il valore dell'attributo **VALUE** ***corrispondente*** all'elemento selezionato.

- Nei menù a tendina (**SELECT**)
  - Viene inviato il testo contenuto tra i tag
  - **<OPTION>** e **</OPTION>** **(***corrispondente alla voce selezionata)*.