



Elaborazioni server- side: dalle CGI al PHP

Cesare Monti 5 febbraio 2003



Cosa vedremo

- HTTP
- Cosa sono le CGI
- Cos'è PHP
- Esempi pratici
- Confronti vari



HTTP: from stateless to interaction



HTTP

HTTP: Web's RPC (Remote Procedure Calls) basate su TCP/IP

Il web viene visto come un grande URL-based file server

Il web su http è centrato sul concetto di Documento e di ipertesto

HTTP è senza stato (stateless)

Ogni richiesta corrisponde ad una nuova connessione

Non c'è interazione tra una richiesta e l'altra



Il protocollo HTTP

- Si basa su due fasi:
 - Richiesta (HTTP:Request)
 - Risposta (HTTP:Response)

Ognuna di queste fasi è a sua volta scomponibile:

Request: line, header, body

Response: header line, headers fields, body



HTTP: Request

- Request line (nome del comando da invocare, es: GET)
- Request header field (informazioni aggiuntive, es: parametri di RPC)
- Entity body (riservato al passaggio di informazioni “bulk” al server)

HTTP:Request

- Sintassi:

<code><method><resource identifier><http version> <clrf></code>	}	Request line
<code>[<Header> : <value>] <clrf></code>	}	Request Header Fields
<code>: :</code>		
<code>: :</code>		
<code>[<Header> : <value>] <clrf></code>	}	Entity Body
<code>blank line <clrf></code>		
<code>[Entity body]</code>	}	

- Esempio:

<code>GET /path/to/file.html HTTP/1.0</code>	}	Request line
<code>Accept: text/html</code>	}	Request Header Fields
<code>Accept: audio/x</code>		
<code>Accept: image/gif</code>		
<code>User-agent: MacWeb</code>	}	



HTTP : Response

- Response Header Line (protocollo e numero di errore)
- Response Header Field (informazioni aggiuntive, contenuto, lunghezza ecc...)
- Entity body (il corpo della pagina richiesta)

HTTP: Response

- Sintassi:

<code><HTTP Version> <result code> [<explanation>] <clrf></code>	}	Response line
<code>[<Header> : <value>] <clrf></code>	}	Response Header Fields
<code>:</code>		
<code>:</code>		
<code>[<Header> : <value>] <clrf></code>		
<code>blank line <clrf></code>		

`[Entity body]`

}

Entity Body

- Esempio:

<code>HTTP/1.0 200 OK</code>	}	Response line
<code>SERVER: NCSA/1.3</code>	}	Response Header Fields
<code>Mime_Version: 1.0</code>		
<code>Content_type: text/html</code>		
<code>Content_length: 2000</code>		
<code><HTML></code>	}	Entity Body
<code>:</code>		
<code></HTML></code>		

HTTP: CGI



- Nate per aggiungere interazione tra client e server
- Il codice risiede interamente sulla macchina server
- Si lascia aperta la possibilità di eseguire codice chiamandolo dal web
- Questa possibilità aderisce ad un protocollo di Common Gateway Interface



CGI : il protocollo

- I clients richiedono l'esecuzione di un programma
- I server invocano il programma chiamato nell'URL di ingresso e utilizzano il protocollo CGI per interpretare il metodo (GET, POST) con cui passare i parametri al programma invocato (via stdin)
- Il programma viene eseguito e ritorna la risposta in formato HTML (via stdout) al server Web
- Il server Web rigira la risposta al client



CGI : Ulteriori migliorie – lo stato

- Tramite CGI è stato introdotto il concetto di stato legato all'HTTP
- Lo stato viene mantenuto attraverso l'invio di variabili che vengono mantenute in memoria dal client (cookies)



CGI: How To

- Possono essere scritte in qualsiasi linguaggio che possa venire interpretato dalla macchina server, quelli più usati sono:
 - C/C++
 - Fortran
 - PERL
 - TCL
 - Any Unix shell
 - Visual Basic
 - AppleScript

CGI: il passaggio dei dati

- GET

viene riscritto run time l'URL della risorsa a cui si vuole accedere aggiungendo i campi che si vogliono passare all'URL stesso

- Es:

- <chiamata a: `http://www.unSito.com/unoScriptCgi.exe`> con parametri:
`param1=10` , `param2= ciao`

Il request line diventa:

```
G E T   w w w . u n S i t o . c o m / u n o S c r i p t C g i . e x e ?  
param1=10&param2=ciao HTTP/1.0
```

CGI: il passaggio dei dati

- POST

- Tutti i parametri vengono passati dentro a Entity Body e viene modificato il method del Request Line

- Es:

<chiamata a: `http://www.unSito.com/unoScriptCgi.exe`> con parametri:
`param1=10 , param2= ciao`

```
POST www.unSito.com/unoScriptCgi.exe HTTP/1.0
```

```
...
```

```
Param1=10
```

```
&param2=ciao
```

CGI: pro e contro dei metodi

- GET

Pro: i parametri rimangono visibili all'utente e l'output della pagina dinamica può essere bookmarcato

Contro: si deve essere sicuri che lo script CGI reso disponibile non possa eseguire azioni dannose al file system a fronte di parametri sbagliati, nella stesura dello script occorre fare parsing sull'URL per avere i parametri



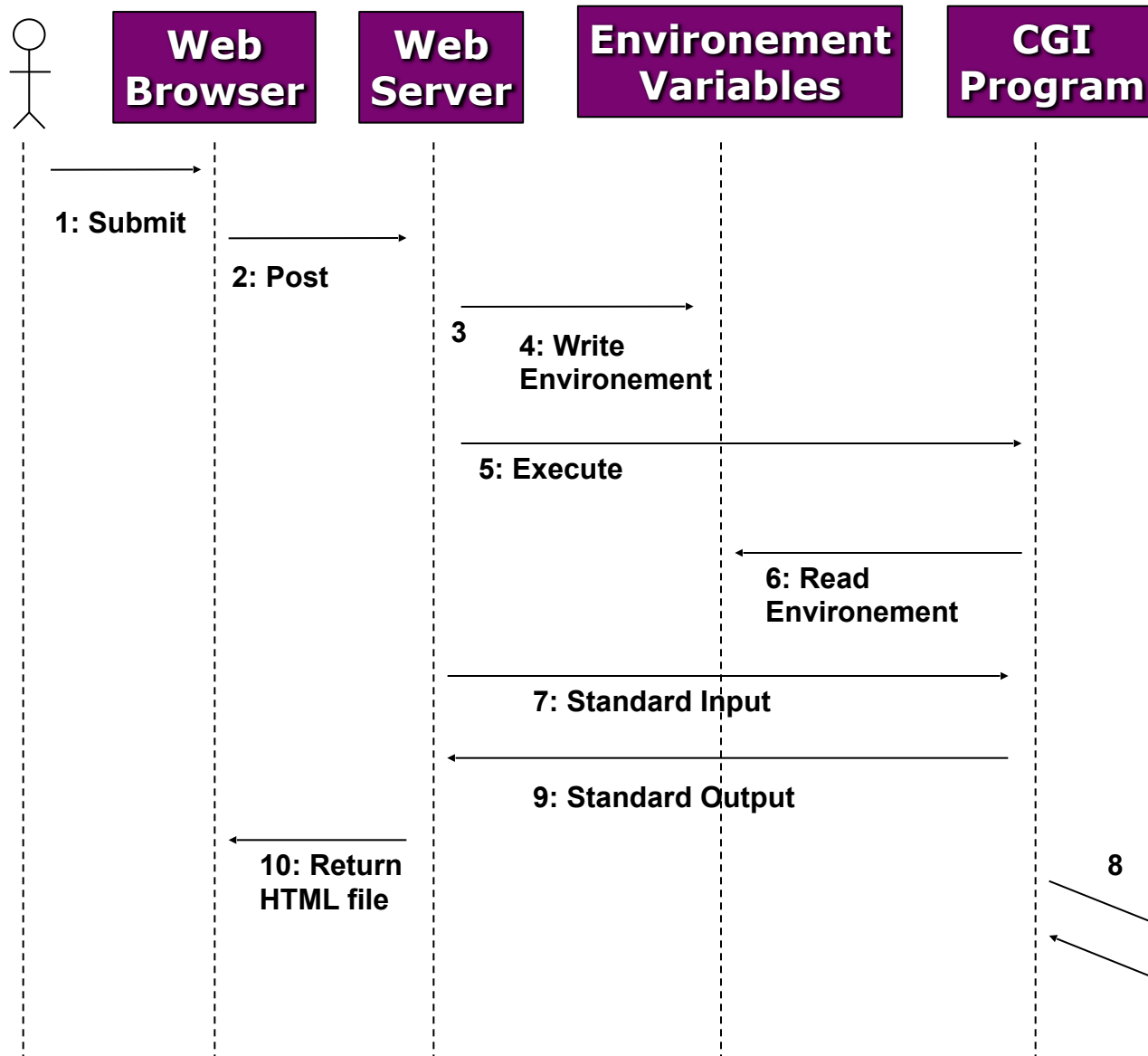
CGI: pro e contro dei metodi

- **POST**

Pro: non occorre dividere URL dai parametri

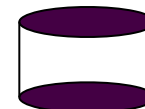
Contro: nel caso di pacchetti incompleti non si può eseguire nulla, le pagine non possono essere bookmarcate

CGI: scenario



1. L'Utente manifesta la volontà di sottomettere dati ad elaborazione, il browser colleziona i dati, controlla il metodo da invocare e genera la request
2. Il browser invoca il metodo POST e sottomette la request
3. Il Server riceve la richiesta e inizia la sessione di risposta
4. Il Server setta le variabili di environment (server name, request method, path_info, content_type ...)
5. Il server HTTP dà inizio al programma CGI
6. Il programma CGI legge le variabili d'ambiente
7. Riceve da stdin i dati utente
8. Fa qualcosa
9. Ritorna i risultati su stdout
10. Il server HTTP formatta il risultato e lo rispedisce al client

8



Example: C code

```
main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES]

    int cl;

    printf("Content-type: text/html%c%c",10,10);
// CHECK SUL CONTENT TYPE
if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-urlencoded")) {
    printf("This script can only be used to decode form results. \n");
    exit(1);
}
    cl = atoi(getenv("CONTENT_LENGTH"));
// AQUISIZIONE DATI DA STDIN
    for(x=0;cl && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fmakeword(stdin,'&',&cl);
entries[x].name = makeword(entries[x].val,'=');
    }
// PREPARAZIONE DATI DI OUTPUT
    printf("<H1>Query Results</H1>");
    printf("You submitted the following name/value pairs:<p>%c",10);
    printf("<ul>%c",10);

    for(x=0; x <= m; x++)
        printf("<li> <code>%s = %s</code>%c",entries[x].name,
            entries[x].val,10);
    printf("</ul>%c",10);
}
```



PHP

Cos'è PHP

- PHP letteralmente è un acronimo ricorsivo per PHP HyperText Preprocessor
- PHP è un linguaggio di scripting general-purpose espressamente costruito per lo sviluppo di applicazioni server-side.
- È un linguaggio che permette di scrivere codice incastonato dentro HTML che interagisce con uno script CGI esterno in maniera dinamica (PHP_interpreter)





PHP:Origini e tipologie del linguaggio

- Nasce nel 1994 come progetto "personale" di Rasmus Lerdorf e la prima versione pubblicamente utilizzabile risale al 1995 con il nome di "Personal Home Page".
- Il resto, è storia: come ogni buon progetto che attira l'attenzione di utilizzatori e sviluppatori, il linguaggio si sviluppa come progetto open-source tanto che, nel 1996, già circa 15.000 siti web lo utilizzano; al momento della release 3 (metà del 1999) il numero di server che utilizzano PHP si è decuplicato.
- Oggi ... non si contano ...

CGI e PHP: fondamentali differenze

- PHP è un linguaggio embedded nel codice HTML delle pagine, e non necessita quindi di ulteriori file esterni per essere eseguito;
- uno script PHP, di fatto, non ha bisogno di installazione.
 - Ogni script CGI deve essere caricato sul server in determinate directory con determinati permessi e via dicendo. Con il PHP questo non succede più: oltre a non aver più bisogno di una directory cgi-bin, essendo il codice inserito direttamente nelle pagine, una volta che la pagina sia caricata di fatto lo script può dirsi pronto per l'uso, a patto che sia stato correttamente configurato;
- con il PHP non si ha più bisogno di particolari configurazioni del webserver in modo da abilitare directory cgi-bin oppure abilitare l'esecuzione di determinati file con determinate estensioni. Una volta istruito il webserver, ogni script (o meglio, pagina con all'interno il codice dello script) potrà essere eseguito in OGNI directory esso si trovi. Immaginate la comodità.



Creazione delle pagine

- Per poter utilizzare un linguaggio occorre predisporre un ambiente di lavoro, nel caso del PHP occorre abilitare il web server a ridirigere la request all'interprete e catturarne il risultato
- Affinché il codice venga eseguito dal server per l'invio al client, poi, è necessario dare alle pagine una determinata estensione e rendere il webserver capace di interpretare tali pagine e richiamare il motore per la loro interpretazione.
- Per Apache, ad esempio, sarà necessario caricare il modulo relativo al PHP nel file di configurazione e definire un'appropriata estensione per le pagine PHP.



Struttura del Linguaggio

1. Sintassi Fondamentale
2. Types
3. Variables
4. Costanti
5. Expressions
6. Operatori
7. Strutture di controllo
8. Funzioni
9. Classi e Oggetti



1) Sintassi fondamentale

- 1.1 Come fare eseguire codice all'interprete PHP
- 1.2 Separazione delle istruzioni
- 1.3 Commenti

Come fare eseguire codice a PHP

...anzitutto si scrive un file php ...

`<? echo ("questo è il più semplice, ovvero come istruzione SGML\n"); ?>`

`<?= espressione ?>` Questa è un'abbreviazione per "`<? echo espressione ?>`"

`<?php echo("se si vogliono produrre documenti XHTML o XML, si utilizzi questo modo\n"); ?>`

`<script language="php">`

`echo ("alcuni editor (tipo FrontPage) non amano le istruzioni di elaborazione");`

`</script>`

`<% echo ("Opzionalmente puoi utilizzare tag nello stile ASP"); %>`

`<%= $variable; # Questo è una abbreviazione per "<%echo .." %>`

Perché co-esistono insieme PHP e HTML ?

```
<?php
if (boolean-expression) {
?> <strong>This is true.</strong>
<?php
} else
{
?> <strong>This is false.</strong>
<?php } ?>
```

questo funziona perché PHP interpreta tutto ciò che trova tra ?> e <? come un'istruzione di echo ();

Separazione delle istruzioni

- Le istruzioni sono separate come nel C o in perl - ogni istruzione termina con un punto e virgola.
- Il tag di chiusura (?>) implica anche la fine di un'istruzione, perciò le seguenti sono equivalenti:

<?php echo "Questo ` un test"; ?>

<?php echo "Questo ` un test" ?>

Commenti

- il PHP (nativo) supporta tre tipi di commenti:

`/* */` -> come nel linguaggio C;

`//` -> come nel linguaggio C++;

`#` -> come nello scripting di shell e nel Perl.

- Questa derivano dal fatto che originariamente il PHP (quando era Personal Home Page) era una procedura che permetteva di richiamare codice C da una pagina web con tag speciali

2) Tipi

- PHP supporta otto diversi tipi primitivi:

scalari	{	1. Boolean
		2. Integer
		3. Floating-point-number (float)
		4. String
		5. Array
composti	{	6. Object
		7. Resource
		8. NULL
speciali	{	



Booleans

- Il tipo boolean esprime la verità, può assumere due soli valori: TRUE | FALSE
- Sono considerati falsi i valori:
 - **FALSE**
 - **L'integer 0**
 - **Il float 0.0**
 - **La stringa "" e la stringa "0"**
 - **Un array con 0 elementi**
 - **Un oggetto con 0 elementi**
 - **Il tipo speciale NULL**
- Tutti gli altri sono considerati veri (compreso -1!)

Integers

- Un integer è un numero appartenente all'insieme di numeri interi $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$
- Un integer può essere specificato in base decimale, ottale o esadecimale
 - \$a = 1234; # integer in base decimale
 - \$a = -123; # integer negativo in base decimale
 - \$a = 0123; # integer in base ottale (equivalente all'83 decimale)
 - \$a = 0x1A; # integer esadecimale (equivalente a 26 decimale)
- Php non supporta unsigned integer
- La dimensione di un integer è platform-dependent (usually 32 bit)
- Se si specifica un integer fuori dai limiti di rappresentazione, questo viene convertito automaticamente in un float

Floating Point Number (float)

- Un float può essere definito con tre diverse sintassi:
 - **\$a = 1.234;**
 - **\$a = 1.2e3; // (equivalente a $1,2 * 10^3 = 1200$)**
 - **\$a = 7E-10; // (equivalente a $1,2 * 10^{-3} = 0.0012$)**
- La dimensione di un float è platform-dependent (solitamente 64 bit con 14 digit di precisione [IEEE standard float])



String

- Una stringa in PHP è una serie di caratteri, un carattere ha le dimensioni di un byte e ci sono 256 configurazioni possibili di caratteri. Non c'è supporto nativo per Unicode
- Esistono tre differenti modi di specificare una stringa:
 - **Single quoted**
 - **Double quoted**
 - **Heredoc syntax**

String (single quoted)

- `echo 'questa è una stringa single quoted';`
- `echo 'Questa è una stringa single quoted con
Terminazione in nuova riga';`
- Gli unici caratteri che PHP interpreta in una single quoted sono `\'` e `\\`
- `echo 'Are you sure you want to delete C:*. *?'; //produce : Are you sure you want to
delete C:*. *?`

String (double quoted)

Double quoted string risolvono molti più caratteri di escape

sequence	meaning
<code>\n</code>	linefeed (LF or 0x0A (10) in ASCII)
<code>\r</code>	carriage return (CR or 0x0D (13) in ASCII)
<code>\t</code>	horizontal tab (HT or 0x09 (9) in ASCII)
<code>\\</code>	backslash
<code>\\$</code>	dollar sign
<code>\"</code>	double-quote
<code>\[0-7]{1,3}</code>	the sequence of characters matching the regular expression is a character in octal notation
<code>\x[0-9A-Fa-f]{1,2}</code>	the sequence of characters matching the regular expression is a character in hexadecimal notation

String (heredoc syntax)

- Heredoc syntax è un formato stringa introdotto recentemente per salvaguardare la compatibilità con sistemi operativi eterogenei
- Sono interpretate come le double quoted ma non presentano il `\r\n` finale tipico dei sistemi Microsoft

```
<?php
$str = <<<EOD Example of string spanning
multiple lines
using heredoc
syntax.
EOD;
?>
```

String as array of char

- È consentito (ma sconsigliato) utilizzare la stringa come una array di caratteri
- È possibile quindi utilizzare una stringa come un vettore indicizzato di caratteri utilizzando l'indice di array tra { }

```
/* Get the first character of a string */  
$str = 'This is a test.'; $first = $str{0};
```

```
/* Get the last character of a string. */  
$str = 'This is still a test.';  
$last = $str{strlen($str)-1};
```

Arrays

- Viene mappato sempre come un array di object,
- L'indicizzazione è libera, di default utilizza integer, ma la mappatura degli indici utilizza stringhe
- La sintassi per l'utilizzo è *nome_array['stringa_indice']*

```
$colore[0]="red";
```

```
$colore[1]="blue";
```

è indifferente scrivere:

```
$colore['red']="red";
```

```
$colore['blue']="blue";
```

Object

- Un oggetto si definisce prima come classe e si istanzia con il costruttore `new`

```
class triangolo {  
  
    function area () {  
        ...  
    }  
}  
  
...  
$triangolo1 = new triangolo;  
triangolo1 --> area();
```



Resource

- È un tipo che referencia una risorsa esterna,
- Di tipo resource sono ad esempio le connessioni con i data base, o il supporto per la connessione ftp o telnet per la creazione di pdf, per la chiamata a classi java, per le socket, per i file ...



NULL

- Introdotta da poco tempo (PHP 4.0) riferisce il tipo di una variabile nulla o di una variabile che riferisce un oggetto distrutto o di una costante al valore NULL

... sorpresa ...

- In PHP non è necessario attribuire un tipo ad ogni variabile, è il PHP stesso che run-time decide il tipo di ogni variabile in funzione del contesto
- Questo è dovuto al concetto di assegnamento e valutazione delle variabili che vedremo poi
- Esistono comunque tecniche di Type Juggling (casting, setType(), getType()) *//molto utili lavorando a classi*)



3) Variabili

- Generiche
- Predefinite
- Variable scope
- Variable ... variables
- Variable from outside PHP

Generiche

- Una variabile è definita in PHP dal \$ seguito dal nome della variabile
- Il nome è case sensitive
- I nomi delle variabili possono cominciare con un carattere o con underscore _ ma non con un numero
- Una variabile punta al suo valore, è possibile copiare il riferimento es:

```
$uno='uno';
```

```
$due=$uno;          //viene copiato il valore della variabile
```

```
$tre=&$uno;          //viene copiato solo il riferimento, un cambiamento alla variabile $uno si  
propaga anche su $tre
```



Predefinite

- ... provate ad utilizzare la funzione `phpinfo()`;
- ...
- <http://sd.ingce.unibo.it/info.php>

Variable scope

- Una variabile definita globale è visibile in tutte le funzioni,
- una variabile definita internamente ad una funzione ha vita solo per la funzione stessa,

```
$a = 1; /* global scope */  
function Test() {  
    echo $a; /* reference to local scope variable */  
}  
Test();
```

Questo script non produce output in quanto \$a utilizzata dentro Test non è definita

Per utilizzare variabili globali dal corpo di funzioni occorre utilizzare lo statement global prima della dichiarazione

```
$a = 1;  
$b = 2;  
function Sum() {  
    global $a, $b;  
    $b = $a + $b;  
}  
Sum();  
echo $b;
```

Variable Variables

- È comodo avere nomi di variabili dinamici, per utilizzarli occorre qualche accorgimento:

```
$a = "hello";
```

```
$$a = "world";
```

```
echo "$a ${$a}"; // Cosa produce in output ?
```

ATTENZIONE:

```
$$a[1] // Array style - produce ambiguità , e non viene interpretato
```

Il parser non saprebbe distinguere tra `${$a[1]}` e `${$a}[1]`



Variables from outside PHP

- Sono le più utilizzate nel mondo PHP
- PHP permette:
 - **Di recuperare valori da form html**
 - **Di settare cookies**
 - **Di recuperare valori di dimensione (altezza e larghezza) da immagini**
 - **Di interagire con variabili esterne (da oggetti java o com)**



Variables from HTML

- Nel momento in cui una form HTML viene spedita, PHP rende disponibile tutti i valori dei campi della form in automatico, creando variabili con il nome del campo form e con il valore a cui sono state settate.
- In altri termini PHP recupera autonomamente le variabili passate (siano esse passate in GET o in POST) rendendole disponibile come variabili globali

Variables from HTML (es)

```
<HTML>
```

```
<body>
```

```
<form name="form1" action="provaAcquisizione.php" method="GET">
```

```
<input type="text" name="campoNome">
```

```
<input type="radio" name="radio1" value="true">
```

```
</form>
```

```
</body>
```

```
</html>
```



Variables from HTML (es)

provaAcquisizione.php

```
<?
```

```
    echo "$campoNome";
```

```
    echo "$radio1";
```

```
?>
```

PHP – una pagina, una funzione?

- L'indipendenza dal metodo di invocazione rende possibile una progettazione funzionale del nostro sistema.
- Svincolandosi dalla chiamata, abbiamo la possibilità di progettare ogni pagina come se fosse una funzione che produce output (in formato HTML) riutilizzabile.
- Es:
 - **prova**
 - **<form method=[POST | GET] action=provavar.php>
 <INPUT TYPE="text" NAME="var1" value="hello ">
 <INPUT TYPE="text" NAME="var2" value="world">
 </form>**

4) **Costanti**

- Le costanti non richiedono il simbolo di dollaro (\$) anteposto al nome;
- Le costanti si possono definire e utilizzare in qualsiasi contesto indipendentemente dalle regole dello spazio dei nomi;
- Le costanti non si possono ridefinire o eliminare una volta che siano state definite;
- Le costanti possono contenere solo valori scalari.

Costanti (es)

```
define("COSTANTE", "Ciao mondo.");
```

```
echo COSTANTE; // stampa "Ciao mondo."
```

```
echo Costante; // stampa "Costante" e genera una notice.
```

- Qualcuno potrebbe obiettare che il comando echo dovrebbe stampare a video la parola Costante ... il problema è che qui mancano i delimitatori di stringa e l'interprete cerca nello spazio dei nomi delle variabili

P.S. una notice è un tipo di errore (tra i più lievi) generato dall'interprete PHP, in questo caso viene generato in quanto l'interprete si accorge che si cerca di stampare la stringa Costante in presenza di una costante di nome COSTANTE

Costanti predefinite

- Molto utili per il debug, sono :
 - `__FILE__`
 - `__LINE__`
 - `PHP_VERSION`
 - `PHP_OS`
 - **`TRUE`**
 - **`FALSE`**
 - **`NULL`**
 - `E_ERROR`
 - `E_WARNING`
 - `E_PARSE`
 - `E_NOTICE`
 - `E_ALL`



Costanti predefinti

```
function comunica_errore($file, $linea, $messaggio) {  
    echo "È avvenuto un errore in $file alla linea $linea: $messaggio.";  
}  
comunica_errore(__FILE__, __LINE__, "Qualcosa è andato storto!");
```

5) Espressioni

Estratto da “PHP Applicazioni WEB” di Tobias Ratschiller e Till Gerken:

“ The simplest yet most accurate way to define an expression is "anything that has a value". “

`$a = 5`

è un espressione e rifacendosi al concetto sopra l'utente non è chiamato a tenere traccia del tipo

N.d.r. Tobias Ratschiller è insieme a Rasmus Lerdorf il creatore di PHP, nonché fondatore di Zend Technologies

Associatività	Operatori
sinistra	,
sinistra	or
sinistra	xor
sinistra	and
destra	print
sinistra	= += -= *= /= .= %= &= = ^= ~= <<= >>=
sinistra	? :
sinistra	
sinistra	&&
sinistra	
sinistra	^
sinistra	&
non associativi	== != === !==
non associativi	< <= > >=
sinistra	<< >>
sinistra	+ - .
sinistra	* / %
destra	! ~ ++ -- (int) (float) (string) (array) (object) @
destra	[
non associativi	new

6) Operatori – precedenza

La seguente tabella fornisce una lista della precedenza degli operatori con gli operatori a più bassa precedenza listati prima.

Operatori Aritmetici

Esempio	Nome	Risultato
$\$a + \b	Addizione	La somma di $\$a$ e $\$b$.
$\$a - \b	Sottrazione	La differenza di $\$a$ e $\$b$.
$\$a * \b	Moltiplicazione	il prodotto di $\$a$ e $\$b$.
$\$a / \b	Divisione	Quoziente di $\$a$ e $\$b$.
$\$a \% \b	Modulo	Il resto di $\$a$ diviso da $\$b$.

Operatori di assegnamento

- L'operatore di base dell'assegnazione è “=” (assegna il valore a)

`$a = ($b = 4) + 5; // $a è uguale a 9 ora, e $b è stato impostato a 4.`

`$a = 3;`

`$a += 5;`

`$b = "Ciao ";`

`$b .= "tutti";`

`echo $b // produce Ciao tutti`

Operatori Bitwise

Esempio	Nome	Risultato
$\$a \& \b	And	Sono impostati ad ON i bit che sono ON sia in \$a che in \$b.
$\$a \b	Or	Sono impostati ad ON i bit che sono ON in \$a oppure in \$b.
$\$a \wedge \b	Xor	Sono impostati ad ON i bit che sono ON in \$a oppure in \$b ma non quelli che sono entrambi ON.
$\sim \$a$	Not	Sono impostati ad ON i bit che sono OFF in \$a, e viceversa.
$\$a \ll \b	Shift left	Sposta i bit di \$a a sinistra di \$b passi (ogni passo significa "moltiplica per due")
$\$a \gg \b	Shift right	Sposta i bit di \$a a destra di \$b passi (ogni passo significa "dividi per due")

Es:

```
echo 12 ^ 9; // L'output è '5'
```

Operatori di Confronto

Esempio	Nome	Risultato
<code>\$a == \$b</code>	Uguale	TRUE se \$a è uguale a \$b.
<code>\$a === \$b</code>	Identico	TRUE se \$a è uguale a \$b, ed essi sono dello stesso tipo. (Solo PHP 4)
<code>\$a != \$b</code>	Diversi	TRUE se \$a è diverso da \$b.
<code>\$a <> \$b</code>	Diversi	TRUE se \$a è diverso da \$b.
<code>\$a !== \$b</code>	Non identici	TRUE se \$a è diverso da \$b, o se essi non sono dello stesso tipo. (Solo PHP 4)
<code>\$a < \$b</code>	Minore	TRUE se \$a è strettamente minore di \$b.
<code>\$a > \$b</code>	Maggiore	TRUE se \$a è strettamente maggiore di \$b.
<code>\$a <= \$b</code>	Minore o uguale	TRUE se \$a è minore o uguale a \$b.
<code>\$a >= \$b</code>	Maggiore o uguale	TRUE se \$a è maggiore o uguale a \$b.

Un altro operatore condizionale è l'operatore "?:" (o trinario), che opera come in C e molti altri linguaggi.

(condizione) ? (espressione2) : (espressione3);

Operatore controllo errore

- PHP supporta un operatore di controllo dell'errore: il carattere at (@). Quando prefisso ad una espressione in PHP, qualunque messaggio di errore che potesse essere generato da quella espressione sarà ignorato.

```
$value = @$array[$indice]; // se $indice non esiste si procederà ugualmente nell'esecuzione
```

```
$my_file = @file ('file_inesistente') or die ("Apertura del file fallita: l'errore è '$php_errormsg'");
```

```
// in questo caso l'esecuzione verrà interrotta ma solo in quanto è stata imposta col comando die()
```

!!! Attenzione all'uso poiché questo inibisce anche i messaggi d'errore dell'interprete PHP!!!

Operatori di esecuzione

- PHP supporta un operatore di esecuzione: backticks (`). (Notare che quelli non sono apostrofi!) PHP cercherà di eseguire il contenuto dei backticks come comando di shell; sarà restituito l'output

```
$output = `ls -al`;  
echo "<pre>$output</pre>";
```

Questo predicato può però essere inibito a livello globale nelle impostazioni dell'interprete PHP

Operatori di incremento/decremento

Esempio	Nome	Effetto
++\$a	Pre-incremento	Incrementa \$a di una unità, inoltre restituisce \$a.
\$a++	Post-incremento	Restituisce \$a, inoltre incrementa \$a di una unità.
--\$a	Pre-decremento	Decrementa \$a di una unità, inoltre restituisce \$a.
\$a--	Post-decremento	Restituisce \$a, inoltre decrementa \$a di una unità.

Operatori logici

Esempio	Nome	Risultato
\$a and \$b	And	TRUE se entrambi \$a e \$b sono TRUE .
\$a or \$b	Or	TRUE se uno tra \$a o \$b è TRUE .
\$a xor \$b	Xor	TRUE se uno tra \$a o \$b è TRUE , ma non entrambi.
! \$a	Not	TRUE se \$a non è TRUE .
\$a && \$b	And	TRUE se entrambi \$a e \$b sono TRUE .
\$a \$b	Or	TRUE se uno tra \$a o \$b è TRUE .

Operatori di stringa

- Ci sono due operatori di stringa:
 - **Concatenazione** .
 - **Assegnazione concatenata** .=

```
$a = "Ciao ";  
$b = $a . "Mondo!"; // ora $b contiene "Ciao Mondo!"
```

```
$a = "Ciao ";  
$a .= "Mondo!"; // ora $a contiene "Ciao Mondo!"
```

7) Strutture di controllo

- Le stesse del C, del Java, e qualcuna del PERL
 - **Esecuzione condizionata (If , else , elseif)**
 - **Esecuzione ciclica condizionata (while, do ... while, for, foreach, ...)**
 - **Terminazione di esecuzione (break, continue, return)**
 - **Controllo alternativo (switch)**
 - **Controllo presenza file di specifiche (require, include, require_once, include_once)**

8) Funzioni

- Una funzione può essere definita usando la seguente sintassi:

```
function esempio ($arg_1, $arg_2, ..., $arg_n) {  
    echo "Funzione di esempio.\n";  
    ...  
    return $retval;  
}
```

- All'interno di una funzione può apparire qualunque codice PHP valido, persino altre funzioni e definizioni di classe.

Funzioni – passaggio argomenti

- Il passaggio degli argomenti può avvenire in due modi:
 - **Per valore**
 - **Per riferimento**

```
function prende_array($input) {                                //per valore
echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

```
function aggiungi_qualcosa(&$string) {    //per riferimento
$string .= 'e qualche altra cosa.';
}
```

Funzioni – passaggio argomenti

- È inoltre possibile e utile specificare valori di default per gli argomenti nel caso non siano stati inizializzati

```
function fare_lo_yogurt ($gusto, $tipo = "yogurt") {  
    return "Fare una vaschetta di $tipo a $gusto.\n";  
}  
echo fare_lo_yogurt ("fragola");
```

ATTENZIONE

```
function fare_lo_yogurt ($tipo = "yogurt", $gusto) {  
    return "Fare una vaschetta di $tipo a $gusto.\n";  
}  
echo fare_lo_yogurt ("fragola");
```

Valori restituiti

- I valori vengono restituiti usando l'istruzione opzionale return.
- Può essere restituito qualsiasi tipo, incluse liste ed oggetti.

```
function return_Array() {  
  return array (0, 1, 2);  
}  
list ($zero, $uno, $due) = return_Array();
```

```
function &restituisce_riferimento() {  
  return $un_riferimento; }  
$nuovo_riferimento =& restituisce_riferimento();
```

Funzioni variabili

- PHP supporta il concetto di funzioni variabili.
- Ciò significa che se un nome di variabile ha le parentesi accodate ad esso, PHP cercherà una funzione con lo stesso nome del valore della variabile, e cercherà di eseguirla.

```
function uno() {  
    echo "In uno()<br>\n";  
}  
function due($arg = "") {  
    echo "In due(); l'argomento era '$arg'.<br>\n";  
}  
$func = 'uno';  
$func();  
$func = 'due';  
$func('test');
```

9) Classi ed oggetti

- In ambito PHP non si parla di OOP
- Una classe in PHP è una collezione di variabili e funzioni che utilizzano queste variabili
- Risultano comode per organizzare meglio il codice e per renderlo intelligibile a terzi
- Ogni variabile di rappresentazione interna della classe, si dichiara preceduta dal costrutto *var*
- Perché una classe possa accedere alle sue variabili di rappresentazione interna, si deve utilizzare *\$this* come riferimento a se stessa

Classi - sintassi

```
<?
class ragazzo {
    var $nome;
    var $cognome;

    function set_nome($name){
        $this-->nome=$name;
    }

    function set_cognome($surname){
        $this-->nome=$surname;
    }

}
?>
```

Classi e costruttori

- In PHP 4, una funzione diventa un costruttore, quando ha lo stesso nome di una classe ed è definita all'interno della classe stessa
- In PHP 3, una funzione si trasforma in un costruttore quando ha lo stesso nome di una classe.

<?

//riferita alla classe di prima

```
function ragazzo (){  
    $this-->nome=NULL;  
    $this-->cognome=NULL;  
}
```

?>

Classi - estensione

- PHP fornisce supporto per l'ereditarietà ma esclusivamente in forma semplice, non supporta l'ereditarietà multipla

<?

```
class studente extends ragazzo{
```

```
    var $studio;
```

```
    function setCorso($corso){
```

```
        $this-->studio=$corso
```

```
    }
```

```
}
```

?>

Ridefinizioni di metodi

- E' possibile ritrovarsi a scrivere classi con codice che si riferisce a variabili e funzioni di classi base. Si utilizza il comando parent per accedere ai metodi della classe base,

```
class A {  
function example() {  
echo "Sono A::example() e fornisco una funzionalità di base.<br>\n"; }  
}  
class B extends A {  
function example() {  
echo "Sono B::example() e fornisco una funzionalità aggiuntiva.<br>\n";  
parent::example(); }  
}  
$b = new B;
```

Classi - staticità

- Può tornare comodo eseguire un metodo di una classe senza doverla istanziare.
- Ciò si può eseguire tramite l'operatore ::

```
class A {  
function example() { echo "Sono la funzione originale A::example().<br>\n"; }  
}  
class B extends A {  
function example() { echo "Sono la funzione ridefinita B::example().<br>\n";  
A::example(); }  
}  
A::example();  
$b = new B;  
$b->example();
```

!!! \$this non può essere usato !!!



Perché PHP

- Perché è comodo
- Perché è intuitivo
- Non richiede troppa configurazione
- ... è molto orientato al web ...
- ... è un linguaggio di scripting
- Ha una grossa base funzionale
- ... per ch  si trova molta roba gi  sviluppata
- Fornisce supporto per la maggioranza dei data base
- Ha meccanismi di sicurezza (le sessioni)
- Si costruisce a classi
- Funziona sulla maggioranza dei server web



Interazioni con i web server

- È prerogativa di PHP quella di fornire supporto per l'interazione con i server web

Se ad esempio lanciassimo la funzione :

```
array = apache_lookup_uri($url);  
Troveremmo dentro $array
```

Interazioni con i web server (2)

- status (che è poi un codice numerico);
- the_request: il tipo di richiesta, il metodo, il file richiesto ed il protocollo utilizzato;
- method: il metodo utilizzato;
- uri: l'uri relativo alla richiesta;
- filename: il nome del file con il path locale;
- path_info: informazioni sul path;
- no_cache;
- no_local_copy;
- allowed;
- sent_bodyct;
- bytes_sent;
- byterange;
- clenght;
- unparsed_uri;
- request_time.

Interazioni con i web server (3)

- Es su \$uri="http://localhost"
- status: 200
the_request: GET /php/prova.php4 HTTP/1.0
method: GET
uri: /php/localhost
filename: /var/www/php/localhost
path_info:
no_cache: 0
no_local_copy: 1
allowed: 0
sent_bodyct: 0
bytes_sent: 0
byterange: 0
clength: 0
unparsed_uri: /php/localhost
request_time: 965150868

Interazioni con i web client

- Lanciando : *\$array = getallheaders();*
- Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en
Connection: Keep-Alive
Host: localhost
Pragma: no-cache
User-Agent: Mozilla/4.72 [en] (X11; I; Linux 2.2.14 i586)

Interazioni con il web

- In generale esistono queste funzioni:
- `apache_child_terminate` — Interrompe il processo apache dopo la presente richiesta
- `apache_lookup_uri` — Esegue una richiesta parziale della URI specificata e restituisce tutte le informazioni
- `apache_note` — Ricava o imposta una variabile nella tabella notes di Apache
- `apache_request_headers` — Estrae tutti gli header della richiesta HTTP
- `apache_response_headers` — Estrae tutti gli header della risposta HTTP
- `apache_setenv` — Imposta una variabile Apache `subprocess_env`
- `ascii2ebcdic` — Traduce una stringa da ASCII a EBCDIC
- `ebcdic2ascii` — Traduce una stringa da string EBCDIC ad ASCII
- `getallheaders` — Estrae tutti gli header della richiesta HTTP
- `virtual` — Esegue una sotto-richiesta Apache



Supporto nativo alla crittografia

- Troviamo supporto per :
 - **DES**
 - **TripleDES**
 - **Blowfish**
 - **3-WAY**
 - **SAFER-SK64**
 - **SAFER-Sk128**
 - **TWOFISH**
 - **TEA**
 - **RC2**
 - **GOST**
 - **RC6**
 - **IDEA**



Supporti vari

- FTP
 - **Pieno supporto per connessioni ftp**
- Networking
 - **Sono molto utili per risolvere nomi DNS, per mappare porte di sistema remote ...**
- Interazioni con sistemi unix
 - **Si può richiedere da remoto di “parlare” direttamente con i processi di un utente...**
- Supporto XML, XSLT, POP3, SMTP, IMAP, calendar, Compressione (ZIP) , generazione PDF, .COM e .NET, funzioni matematiche, parser, JAVA , TELNET, SNMP ...



PHP handling session

- Come abbiamo già detto le CGI hanno introdotto il concetto di sessione applicato al http
- In PHP le sessioni sono meccanismi per la sincronizzazione e il mantenimento dello stato passando da una pagina all'altra
- Servono per mantenere dati tra un accesso e l'altro
- Tecnicamente la sessione PHP è rappresentata da un file memorizzato sul server dove vengono memorizzate gli stati di tutte le variabili che sono attive nel contesto di una connessione
- Cambiando pagina le variabili rimangono così attive e possono esservene aggiunte altre



PHP - sessioni

- Essendo un meccanismo server side deve essere configurato l'interprete per abilitarne il supporto
- Occorre però gestirle per evitare garbage sulla macchina server
- Vengono tipicamente utilizzate per gestire ambiti in cui è richiesta sicurezza e dove non si voglia ricorrere all'emissione di Cookies (che ricordiamo essere immagazzinati client side)



PHP - Sessioni

Generazione dell'ID di sessione

Esistono vari modi, il più utilizzato è:

```
srand( (double)microtime() * 1000000 );  
$sessionId = md5(uniqid(rand()));
```

La prima riga genera un numero random utilizzando l'orologio di sistema con una precisione di un double

La seconda assume il numero random generato come ID di sessione codificandolo in chiave md5



PHP - Sessioni

- Una volta creato l'ID di sessione, nel momento in cui si voglia proteggere dati sotto sessione occorre segnalarlo all'interprete utilizzando la funzione `session_start()`;
- Per inserire una nuova variabile si usa `session_register()`;
- Una volta registrato un valore di una variabile è recuperabile da qualsiasi applicazione che abbia la stessa sessione fino a che non si distrugge la sessione
- Per distruggere la sessione (operazione consigliata alla fine per eliminare garbage sul server) si usa `session_destroy()`;



PHP - Sessioni

- Nel caso si utilizzino oggetti, questi possono essere reinterpretati dalla sessione solo se l'oggetto è stato serializzato (`serialize();`)
- Per serializzare un oggetto occorre che la definizione di classe sia presente sia nella pagina di partenza che in quella di arrivo
- In PHP 4 la serializzazione degli oggetti passati in sessione avviene in automatico, ma è sempre buona norma includere lo stesso i file con le definizioni di classe

PHP esempi

```
function connectMy (){  
//-----  
$dbUsername = "root";  
$dbPassword = "preferita";  
$dbHostname = "localhost";  
$dbName = "DatabaseName";  
$msg1 = "<b>ERRORE</b>. Ci sono problemi sul server.<br>\n";  
$msg2 = "<b>ERRORE</b>. Ci sono problemi nel database.<br>\n";
```

```
mysql_connect($dbHostname, $dbUsername, $dbPassword) or die ($msg1);  
mysql_select_db($dbName) or die ($msg2);  
}
```

`$connessione=connectMy();` // Cosa ritorna questa funzione ? Come posso riutilizzarla ?



Php Esempi:

```
function generate_session_id () {  
    //-----  
    srand( (double)microtime() * 1000000 );  
    $sessionId = md5(uniqid(rand()));  
    session_id($sessionId);  
}
```

```
//una volta chiamata questa funzione posso far partire le sessioni direttamente con  
session_start();
```

HTML – PHP esempi

```
<html><head><title>Prova PHP – HTML</title></head>
<body>
<form method="POST" action="filephp.php">
<table width="80%" align="center" border="1">
  <tr>
    <td>Login</td>
    <td><input type="text" name="login"></td>
    <td>Password</td>
    <td><input type="password" name="pw"></td>
    <td><input type="submit" name="Submit" value="Invia"></td>
  </tr>
</table>
</form>
</body>
</html>
```

HTML – PHP esempi

```
srand( (double)microtime() * 1000000 );
$sessionId = md5(uniqid(rand()));
session_id($sessionId);
session_start();
require ("./funzioni.php");
$sql="SELECT utenti.* FROM utenti where (utenti.username='".$login. "')";
$connessione=connectMy ();
$result = mysql_query ($sql) or die ("Invalid query");
$arrayResult=mysql_fetch_array($result);
if ($arrayResult["pw"]== $pw){//accesso assicurato abilito l'utente alle funzioni
}
else {session_destroy();}
mysql_close($connessione);
```



PHP – Esempi costruzioni a classi

- Esempio tangibile sono le classi d'interfaccia con i database
- <http://sd.ingce.unibo.it/download/>



PHP – consigli tattici

- Mappare ciò che si vuol fare
- Scomporre il tutto in blocchi funzionali o classi
- Utilizzare gli include/require e riutilizzare il possibile
- Sfruttare al massimo le sessioni



PHP Vs PERL

- In php abbiamo una netta separazione dei due codici
- In PERL abbiamo la piena palmarità del sistema (con molti template che ci aiutano)
- PHP è più performante (sia nell'interpretazione che nel thread)
- PHP è più modulare (organizzando in classi il codice)
- In PHP l'accesso alle variabili è indipendente dal metodo di passaggio, in PERL no



PHP Vs ASP

- Il codice risulta meglio organizzato in PHP
 - **in quanto le variabili HTTP passate in POST e in GET sono viste come globali e non si perde tempo ad estrarle dal REQUEST**
- Il Php ha la funzione include() molto comoda mentre è assente in ASP
- La manipolazione dell'Header della pagina HTTP è meglio gestibile da ASP
- Tutte le funzionalità degli oggetti ASP sono disponibili solamente nei web server Microsoft