

Unità **A1**

Il linguaggio PHP



ESERCIZIO COMMENTATO

La progettazione concettuale: il modello ER



FILE SORGENTI

In questa unità imparerai...

- Che cos'è un server web
- Le basi della programmazione lato server in PHP
- Come inviare le informazioni dal client al server
- Come gestire i contenuti e la formattazione di una pagina web dinamica

1 Programmazione lato client e lato server

Dato un ambiente client/server in un'architettura di protocolli TCP/IP, la **programmazione lato client** è lo sviluppo di applicativi che sono eseguiti prevalentemente sul client, inviando eventuali richieste al server e gestendo i risultati ricevuti da quest'ultimo.

Per **programmazione lato server** intendiamo lo sviluppo di programmi applicativi che sono eseguiti prevalentemente sul server, accettando richieste dal client e fornendo a quest'ultimo i risultati dell'elaborazione in forma di documenti (o pagine) HTML.

Sono esempi di programmazione lato client gli *applet*, il *codice JavaScript* e il *codice VBScript*. Possiamo quindi parlare di **programma lato server** e **programma lato client**, così come di **linguaggi lato server** e di **linguaggi lato client**.

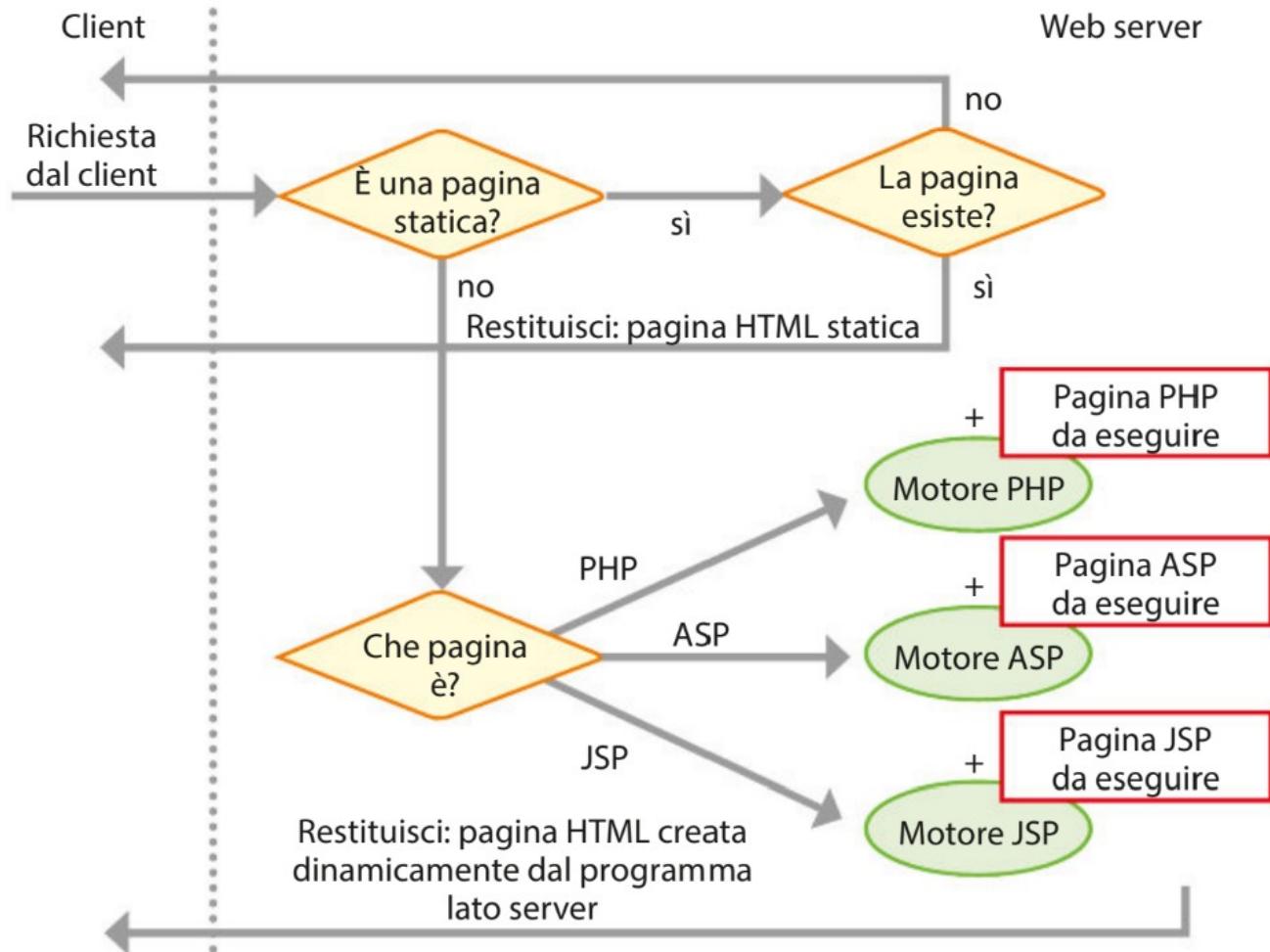
Il web server, infatti, quando riceve una richiesta da parte del client può interpretarla come:

- *semplice richiesta di invio* di pagine HTML statiche presenti sul server (o altre risorse come, ad esempio, un'immagine, un file audio, un applet, e così via);
- *richiesta di esecuzione* di un file contenente le istruzioni del programma lato server.

Nella figura a pagina seguente sono riportate le azioni che vengono intraprese dal web server.

Possiamo notare che, a seguito di un'elaborazione lato server, viene restituita al client una pagina HTML creata dinamicamente dal programma lato server.

Definiamo **programmazione orientata al web** l'insieme di tecniche e metodologie che si utilizzano in un ambiente client/server con un'architettura TCP/IP per far interagire programmi lato server e programmi lato client, con l'obiettivo di realizzare sistemi che possano essere eseguiti in intranet e Internet.



Linguaggi di programmazione e di scripting lato server

Finora abbiamo parlato abbastanza genericamente di **linguaggi di programmazione lato server**, ma spesso occorre distinguere fra:

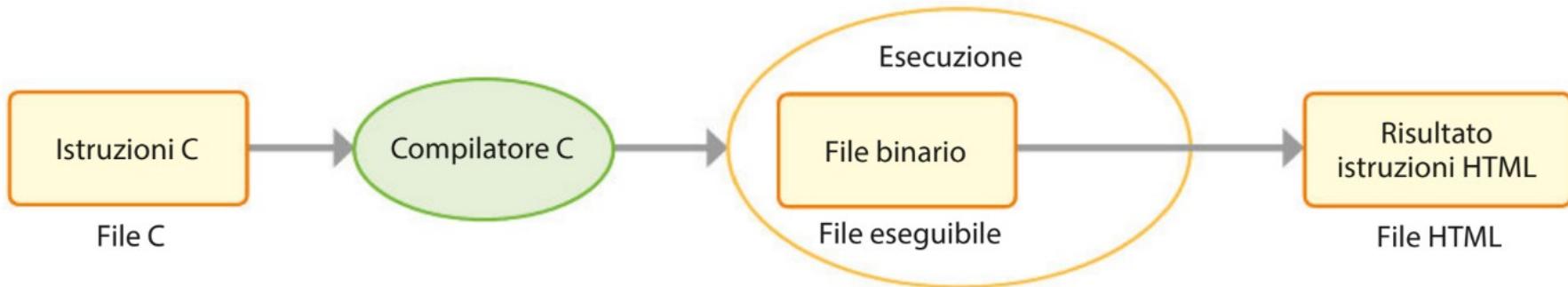
- **linguaggi di programmazione** (veri e propri) lato server; ad esempio Java, con i servlet, oppure C, in cui si scrivono i programmi CGI (uno dei primi metodi di programmazione lato server);
- **linguaggi di scripting** lato server; ad esempio PHP, PERL (particolarmente efficiente nel trattamento delle stringhe), ASP; solitamente sono linguaggi interpretati.

Una differenza consiste nel fatto che Java, ad esempio, è un linguaggio di programmazione che ha vita autonoma anche in versione non lato server. PHP, invece, è un linguaggio che può essere utilizzato solo per applicazioni lato server.

Un programma **CGI** scritto in linguaggio C è un classico esempio di **programma lato server compilato**, il cui eseguibile è inserito in un'opportuna directory sul web server, come mostrato nella figura seguente.

Ricorda

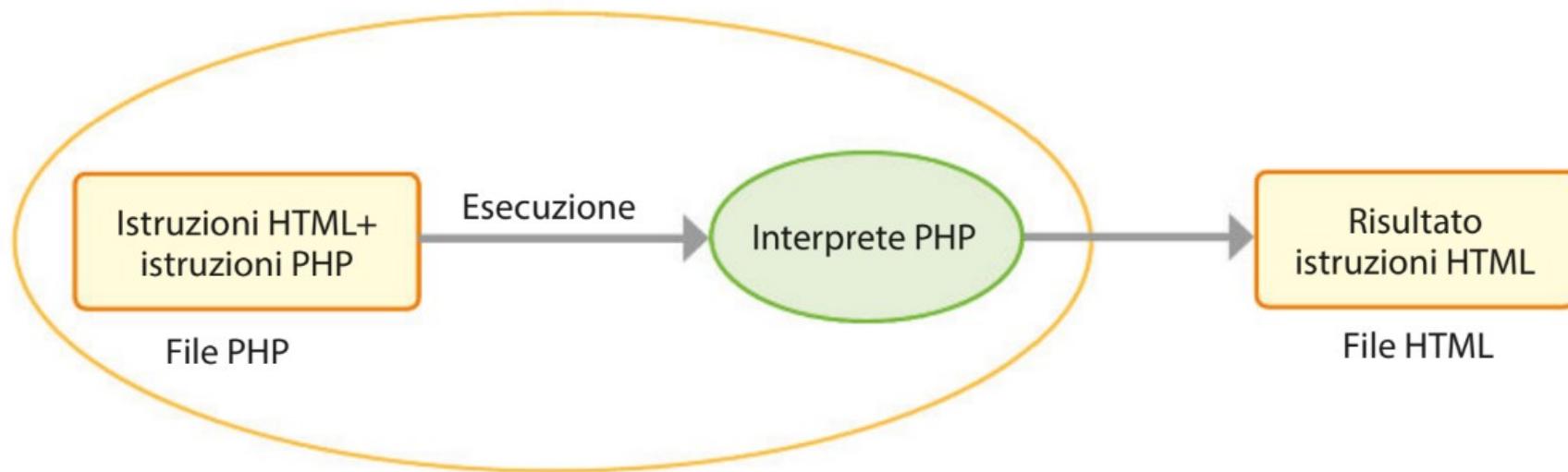
In informatica **Common Gateway Interface** (acronimo **CGI**; in italiano: interfaccia comune, nel senso di standard, per gateway) è una tecnologia standard usata dai web server per interfacciarsi con applicazioni esterne generando contenuti web dinamici.



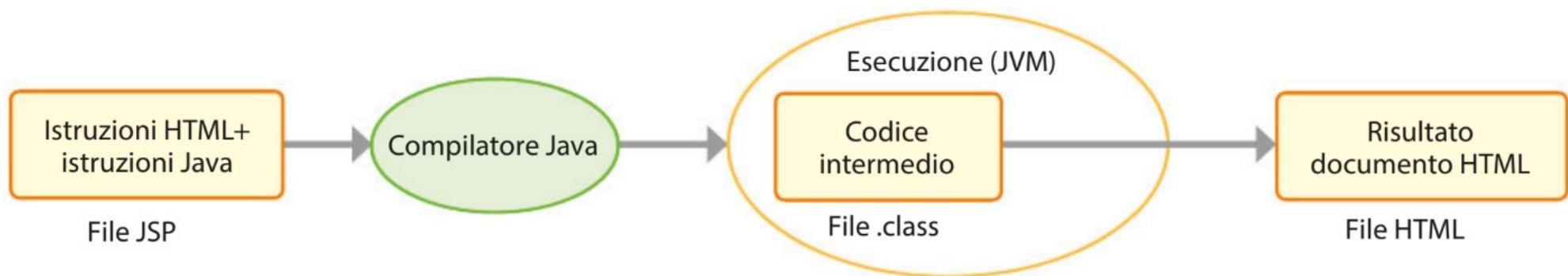
Un file di comandi **PHP** è, invece, un esempio di **programma lato server interpretato**. Il web server, infatti, associa a tale file l'interprete PHP, che deve essere avviato per poterne eseguire i comandi.

Per questo motivo è possibile avere comandi PHP all'interno di pagine HTML oppure pagine PHP pure.

Schematizziamo quanto detto nella figura seguente:



Un approccio misto è quello delle **JSP Java**. Il codice Java deve essere prima compilato, ottenendo un codice intermedio (i file .class), e poi interpretato dall'interprete Java (la Java Virtual Machine) per essere eseguito.



2 Server web

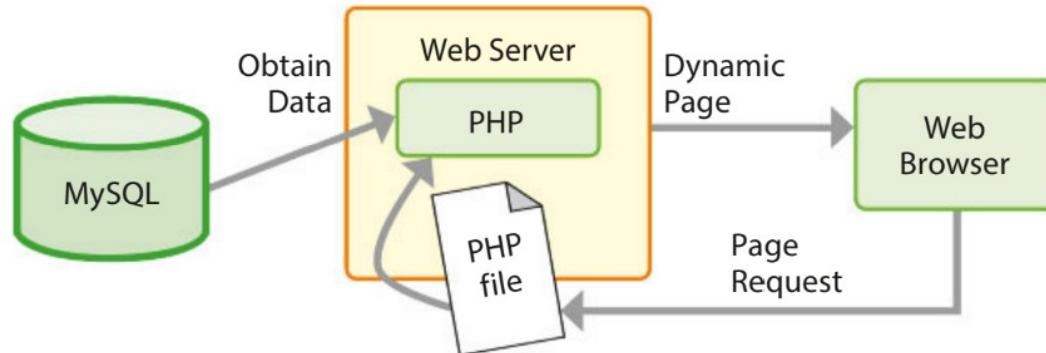
Ricorda

Apache è la piattaforma server web modulare più diffusa, in grado di lavorare su differenti sistemi operativi.

L'esecuzione di uno script in PHP richiede necessariamente un server web che implementi i moduli per la gestione del linguaggio. Uno dei più diffusi server web è **Apache**, un server web libero, sviluppato dalla Apache Software Foundation.

Le alternative per realizzare e provare i programmi in PHP sono essenzialmente due: utilizzare una piattaforma di hosting sulla quale appoggiare i nostri file php o installare sul nostro pc un web server vero e proprio. Se la prima ipotesi consiste nel ricercare una soluzione hosting con le caratteristiche richieste, la seconda ipotesi consente di gestire all'interno del nostro pc tutto quello di cui abbiamo bisogno, indipendentemente dalla disponibilità di una connessione Internet. Il nostro computer, quindi, fungerà sia da client per effettuare delle richieste, sia da server per cercare di soddisfarle.

Oltre un server web, nel prosieguo di questa unità, avremo bisogno anche di un server per gestire i database; pertanto l'attuale obiettivo è quello di installare un pacchetto che contenga già da ora quanto richiesto per le nostre prove.



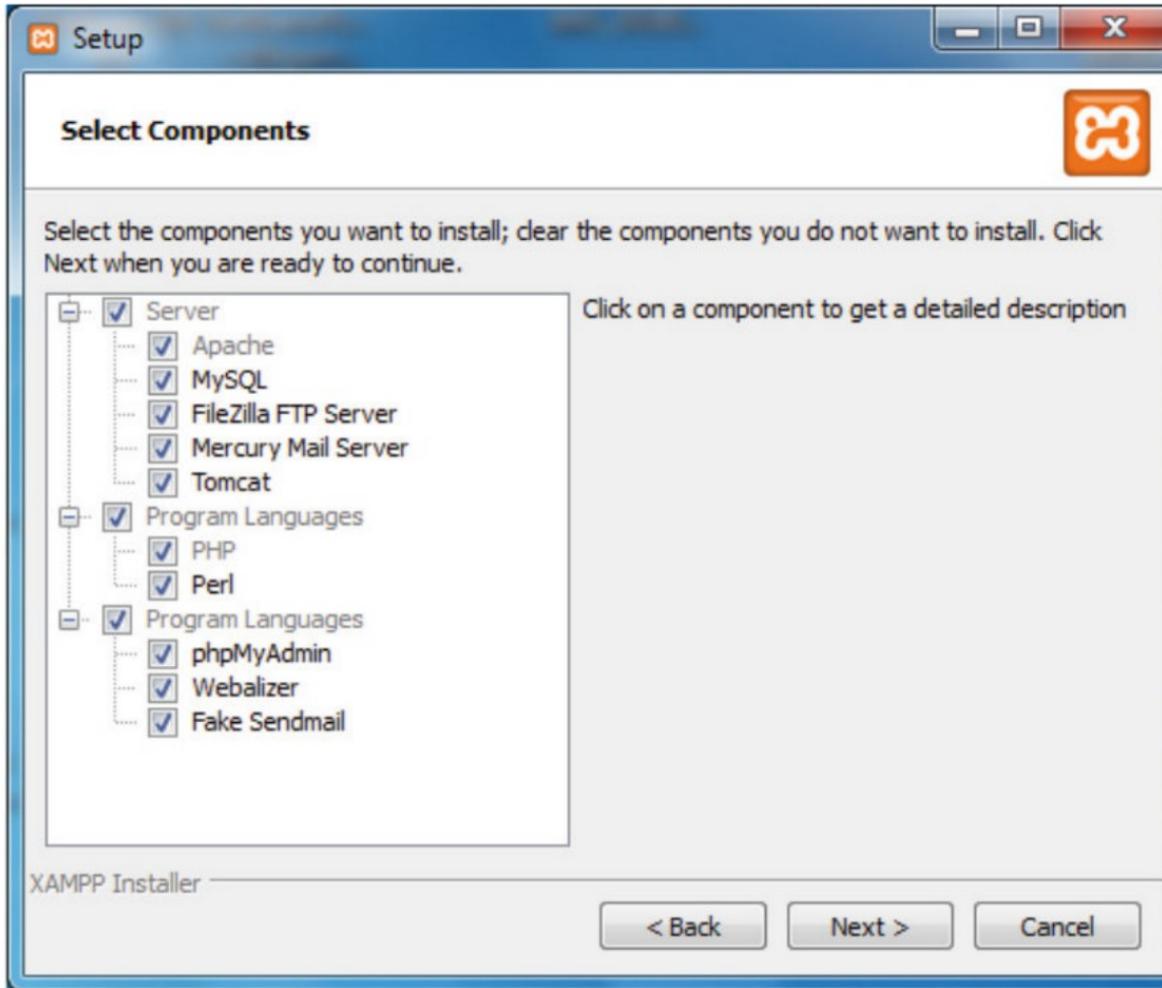
Fortunatamente esistono dei pacchetti già confezionati che ci permettono di installare comodamente, in pochi step, quello che a noi accorre, evitando di dover successivamente assemblare le varie funzionalità. Fra i più consolidati ed efficienti citiamo:

EasyPHP	WAMP	XAMPP
 The logo for EasyPHP consists of three black geometric shapes: a large square at the top left, a smaller square below it, and a trapezoid shape to its right.	 The logo for WAMP features a stylized white letter 'W' inside a pink circle with a white outline.	 The logo for XAMPP is a white lowercase letter 'x' inside an orange rounded square.

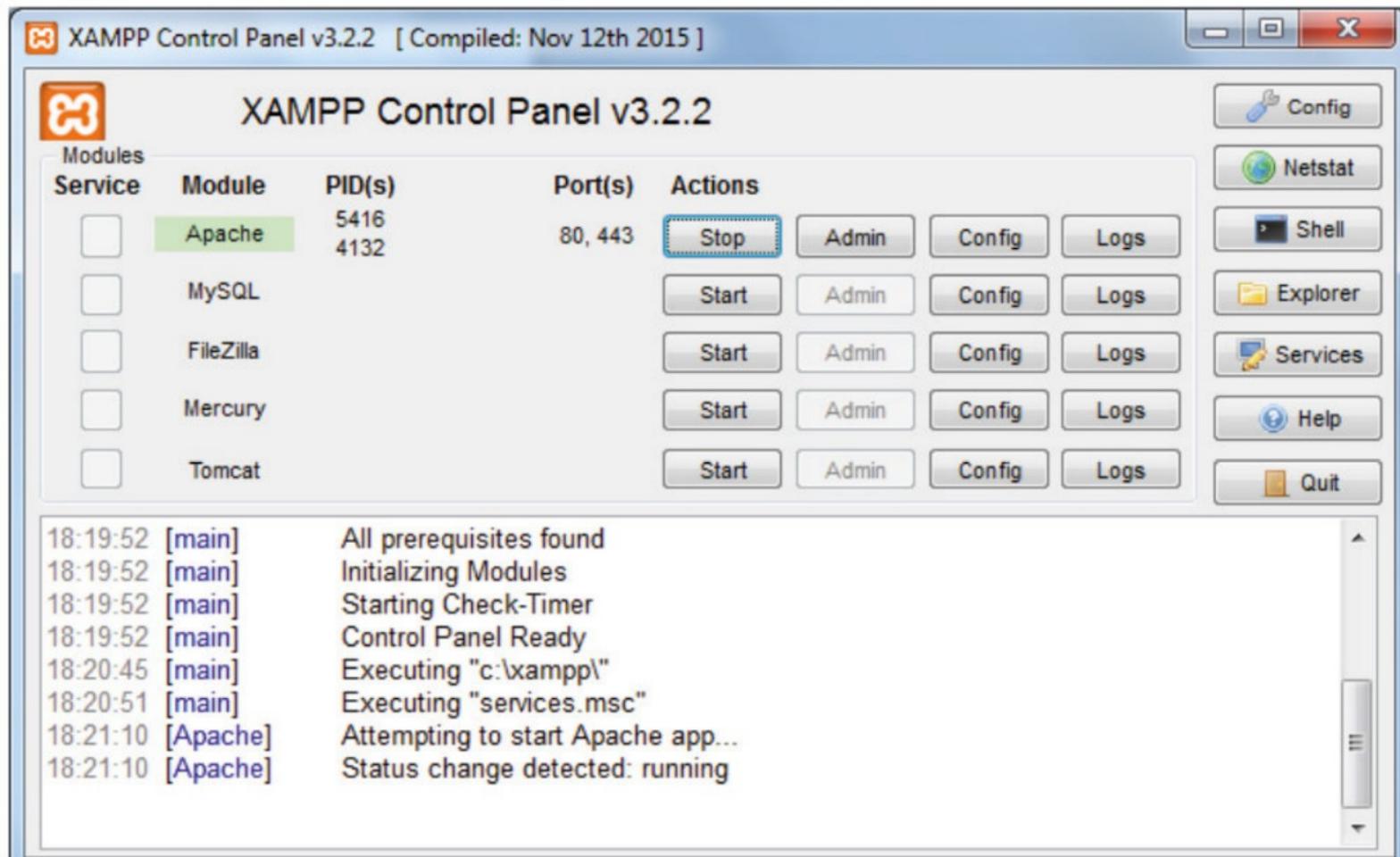
I primi due offrono i medesimi servizi e caratteristiche, un server web (Apache), un server database (MySQL) e il linguaggio PHP di base per accedere a entrambi.

XAMPP offre in più ulteriori servizi, la possibilità di gestire un server per il trasferimento dei file (FileZilla), un server mail (Mercury) e un server web per Java (Tomcat). Pertanto, visto il maggior numero di servizi forniti, sceglieremo di installare quest'ultimo.

Scaricato il pacchetto, la fase di installazione è piuttosto semplice. Una schermata ci riporta i componenti da installare (lasciamo la spunta su tutti, alcuni di essi ci serviranno in seguito).



Terminata la fase di installazione ci viene riproposto il pannello di controllo da cui attivare o disattivare i vari servizi; per il momento è importante fare click su Start in corrispondenza della voce Apache per attivare il server web.



Avviato il server web non resta che verificarne il funzionamento attraverso il client web (browser), semplicemente digitando nella barra degli indirizzi il termine **localhost** o l'indirizzo di loopback **127.0.0.1**

The screenshot shows a web browser window with the URL `127.0.0.1/dashboard/` in the address bar. The page has a dark blue header with the "Apache Friends" logo and links for Applications, FAQs, HOW-TO Guides, PHPInfo, and phpMyAdmin. The main content area features the XAMPP logo (an orange square with a white "X") and the text "XAMPP Apache + MariaDB + PHP + Perl". Below this, a large heading says "Welcome to XAMPP for Windows 7.2.6". A paragraph explains that the user has successfully installed XAMPP and can now start using Apache, MariaDB, PHP, and other components. It also encourages them to check the FAQ and HOW-TO Guides. Another paragraph cautions that XAMPP is meant for development purposes only and may be insecure if used on the internet, suggesting WAMP, MAMP, or LAMP as better options for production environments. At the bottom, there is a link to start the XAMPP Control Panel.

① 127.0.0.1/dashboard/

Apache Friends

Applications FAQs HOW-TO Guides PHPInfo phpMyAdmin

 **XAMPP** Apache + MariaDB + PHP + Perl

Welcome to XAMPP for Windows 7.2.6

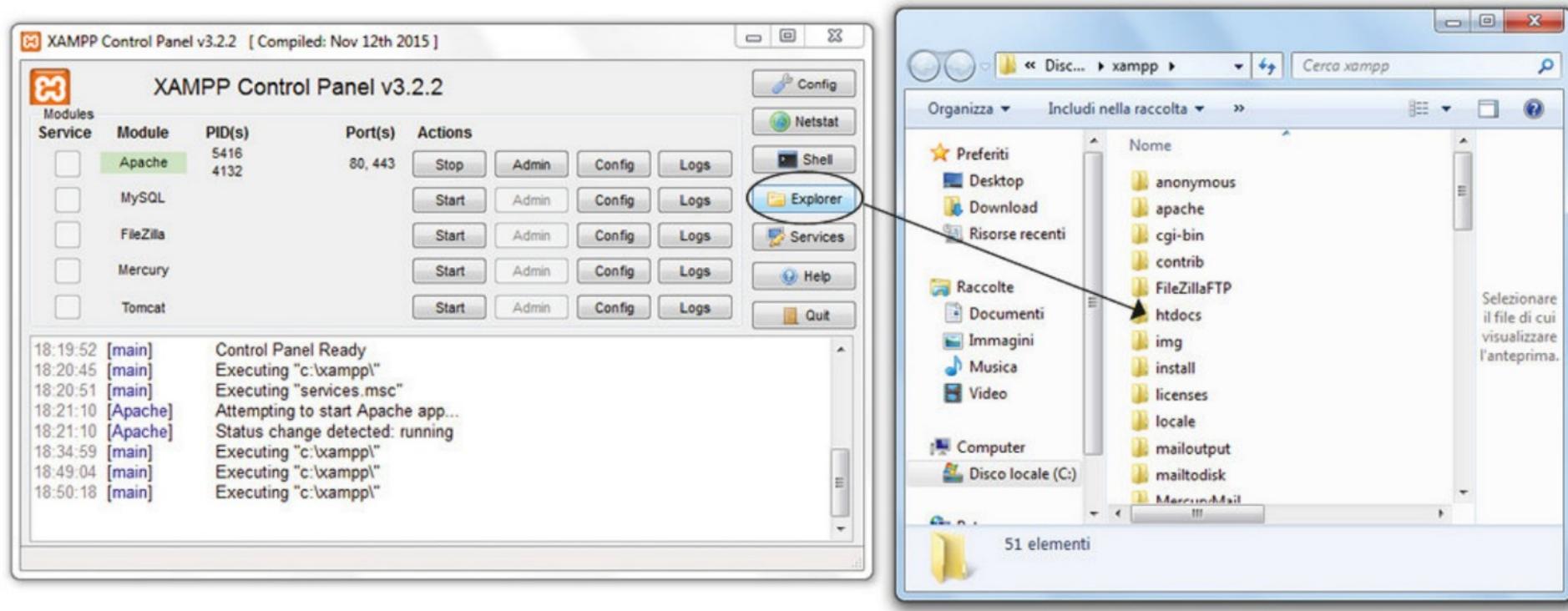
You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others. If you want have your XAMPP accessible from the internet, make sure you understand the implications and you checked the [FAQs](#) to learn how to protect your site. Alternatively you can use [WAMP](#), [MAMP](#) or [LAMP](#) which are similar packages which are more suitable for production.

Start the XAMPP Control Panel to check the server status.

Il server web risponde visualizzando i file contenuti all'interno della cartella **htdocs**; pertanto è necessario posizionarsi all'interno di tale percorso e creare le proprie cartelle in cui inserire i file PHP da testare.

È possibile raggiungere la cartella anche dal pulsante **Explorer** del pannello di controllo di XAMPP.



Se supponiamo di provare il file *Primo.php* posto all'interno di una nostra cartella denominata *FilePersonali*, dovremo digitare nella barra degli indirizzi del browser il percorso:

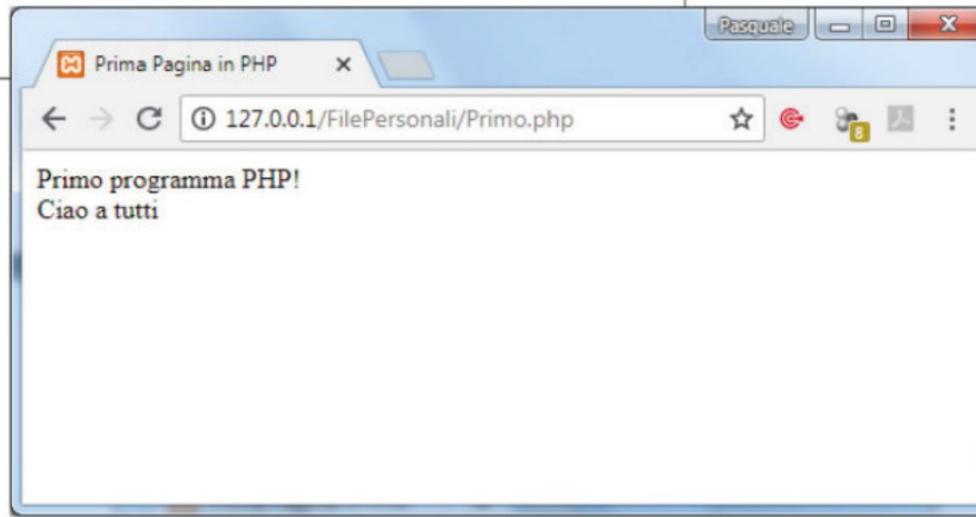
127.0.0.1/FilePersonali/Primo.php

o, in alternativa:

localhost/FilePersonali/Primo.php

Come primo programma riportiamo il codice del file *Primo.php* per iniziare a prendere dimestichezza con il linguaggio PHP all'interno del codice HTML:

```
<html>
    <head>
        <title>Prima Pagina in PHP</title>
    </head>
    <body>
        <?php
            print("Primo programma PHP!<br/>");
            echo("Ciao a tutti");
        ?>
    </body>
</html>
```



3 Istruzioni di output

Nel codice precedente abbiamo utilizzato due istruzioni che possiamo considerare equivalenti e che servono per produrre l'output visualizzato dal browser. Tali istruzioni sono: *print()* ed *echo()*.

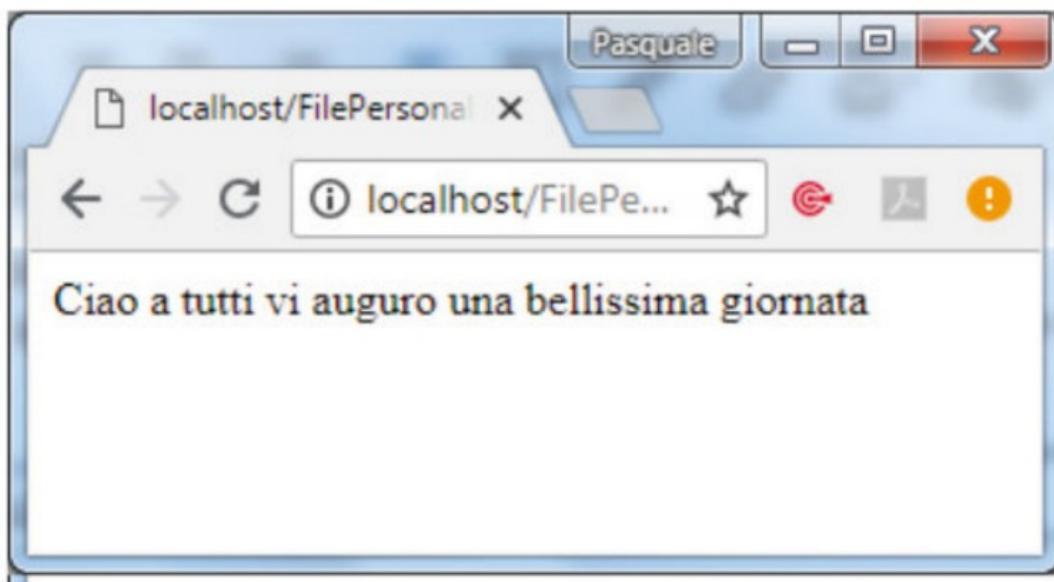
La sintassi di **echo()** è:

```
echo(<Stringa>);
```

Viene visualizzata la <Stringa> posta tra parentesi tonde (che sono opzionali). Nella versione senza parentesi tonde è possibile utilizzare anche la seguente sintassi:

```
echo <Stringa1>, <Stringa2>, ... , <StringaN>;
```

Ad esempio:



```
echo "Ciao a tutti", "vi auguro una  
bellissima giornata";
```

produce quanto è mostrato nella figura a lato.
La sintassi di *print()* è molto simile a quella di *echo()*:

```
print(<Stringa>);
```

oppure:

```
print <Stringa>;
```

A differenza di *echo()*, *print()* accetta un solo parametro in input e restituisce 1 se la visualizzazione ha avuto successo, 0 altrimenti. Nelle stringhe degli esempi precedenti abbiamo utilizzato l'elemento html *
* che permette di andare a capo. L'operatore di concatenazione di PHP è il punto; quindi, si ha lo stesso output sia con *print("ciao". "mondo")*, sia con *echo("ciao", "mondo")*.

Ogni istruzione PHP **deve obbligatoriamente terminare con il punto e virgola**.

4 Variabili: tipi e valori

PHP, seguendo le convenzioni adottate in molti linguaggi di scripting (come ad esempio JavaScript), prevede un controllo non rigido sui tipi delle variabili. Vediamo che cosa implica ciò nella gestione delle variabili definite dall'utente.

Dichiarazione, tipi e assegnazione di valori

In PHP non è necessaria alcuna dichiarazione di variabile e i tipi delle variabili vengono associati con l'assegnazione di valori.

Ad esempio, possiamo definire la variabile X di tipo intero e assegnarle il valore 10 nel seguente modo:

```
$X = 10;
```

Osserviamo che i nomi delle variabili devono essere preceduti dal simbolo del dollaro (\$).

Osserviamo inoltre che l'operatore di assegnazione è “ = ” e che **il tipo della variabile X viene dedotto dal valore assegnato**; in questo caso X è di tipo intero.

La riassegnazione di un valore di un altro tipo è perfettamente legittima. Ad esempio:

```
$X = "nuova stringa";
print($X);
```

visualizza quanto è mostrato nella figura a lato.

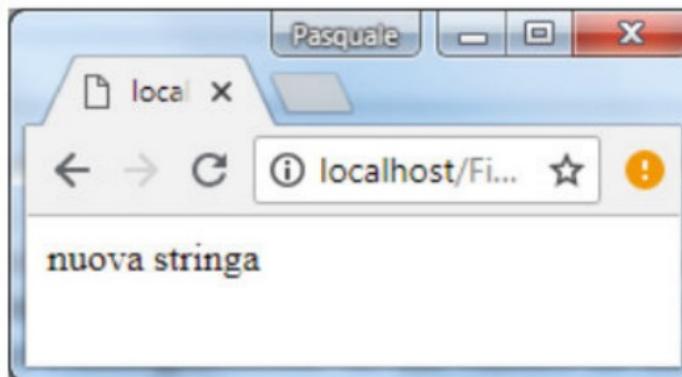
In PHP esistono alcune semplici **regole di nomenclatura** per le variabili:

- il primo carattere deve essere ‘\$’;
- possiamo scegliere il nome delle variabili usando lettere, numeri e il trattino di sottolineatura, o *underscore* (_);
- il primo carattere del nome (subito dopo il \$) deve essere una lettera o un underscore, non un numero.

Occorre inoltre ricordare che il nome delle variabili è *case sensitive*, cioè sensibile alla distinzione tra maiuscole e minuscole: di conseguenza, se scriviamo due volte un nome di variabile usando le maiuscole in maniera differente, per PHP si tratterà di due variabili distinte.

Rifletti

La tipizzazione delle variabili non prevista in PHP potrebbe portare ad errori dovuti all'utilizzo di variabili con lo stesso nome per contesti diversi. Un linguaggio tipizzato non permetterebbe il passaggio di valori differenti dal tipo per il quale la variabile è stata dichiarata e ciò, per i programmati neofiti, potrebbe essere un vantaggio, in quanto il sistema segnalerebbe l'errore.



Valori predefiniti ed errori di tipo *notice*

Poiché le variabili non hanno tipi espressamente dichiarati, il sistema non sa in anticipo se una variabile verrà utilizzata per archiviare un intero oppure una stringa di caratteri. Pertanto il valore predefinito di una variabile **non ancora assegnata** viene interpretato a seconda del contesto. Questo significa che in un contesto che richiede una variabile come un numero, una variabile non assegnata verrà valutata come 0, in un altro contesto che richiede un valore stringa, la variabile non assegnata sarà valutata come la stringa vuota (stringa lunga zero caratteri).

Consideriamo ora il seguente codice PHP:

```
$X = 10;  
$Y = 400;  
$Risultato = $X * $Y;
```

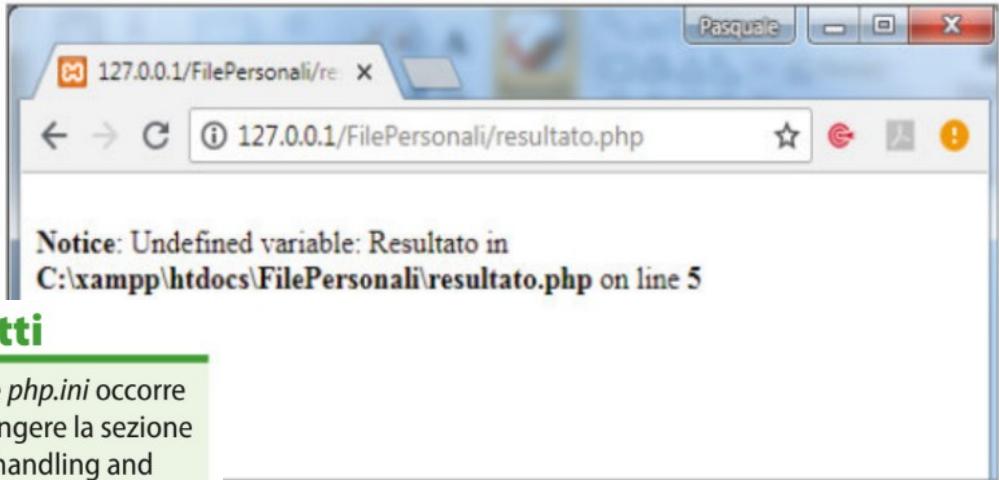
Dopo l'esecuzione del codice precedente, la variabile *Risultato* assumerà il valore 4000.

In realtà, possiamo riferirci a una variabile anche senza che sia stata inizializzata. Ad esempio, supponendo che nel nostro script non sia stato assegnato alcun valore alla variabile *W*, potremmo avere un'istruzione di questo genere:

```
print($W);
```

Questo codice non produrrà alcun output, in quanto la variabile *W* non esiste. Nonostante lo script funzioni regolarmente e proceda con l'elaborazione delle istruzioni successive, un'istruzione di questo tipo non viene considerata corretta da PHP, che produrrà una segnalazione di errore di tipo *notice* per farci notare che abbiamo usato una variabile non inizializzata. Gli errori di tipo *notice* sono quelli di livello più basso, cioè meno gravi, che normalmente non vengono mostrati da PHP, ma di cui possiamo abilitare la visualizzazione attraverso il file di configurazione *php.ini*.

Vediamo un esempio in cui è evidente la necessità di segnalazioni di errori di tipo *notice*. Consideriamo il seguente codice:



```
$X = 10;  
$Y = 40;  
$Risultato = $X + $Y;  
print($Resultato); // Errore;  
abbiamo scritto Resultato al posto di  
Risultato!!!
```

Questo codice vorrebbe assegnare il valore 50 alla variabile *Risultato* e poi stamparne il valore. Però contiene un errore: nell'istruzione di stampa abbiamo indicato la variabile *Resultato* inve-

ce di *Risultato*. Chi ha scritto il codice si aspetterebbe di vedere comparire sul browser il risultato 50, invece non troverà nulla, perché la variabile *Resultato* non è definita, e quindi non ha nessun valore.

Rifletti

Nel file *php.ini* occorre raggiungere la sezione "error handling and logging" e impostare la variabile di sistema *error_reporting*. Tale variabile rappresenta il "livello" degli errori che vengono mostrati da PHP. È consigliabile utilizzare l'impostazione "E_ALL", che mostra tutti gli errori, compresi quelli meno gravi di tipo *notice* (come nella figura sopra).

5 Espressioni

Gli operandi

Gli **operandi costanti** (o **valori letterali**) sono quantità esplicite, il cui tipo non va dichiarato; ad esempio, il valore 45 è esplicitamente considerato numerico.

Tali operandi costanti sono presenti in numero esiguo e sono:

- valori **interi**: rappresentano i numeri senza parte decimale, positivi e negativi, ad esempio 345;
- valori **reali** (o **doppi** o **double**): rappresentano i numeri con parte decimale, ad esempio 345,67;
- valori **booleani**: rappresentano due soli valori: TRUE (vero), FALSE (falso);
- valori **stringa**: rappresentano sequenze di caratteri.

Valori interi

La dimensione degli interi dipende dalla piattaforma hardware, ma generalmente:

- l'intero più grande è $2^{31} - 1$ (2.147.483.647);
- l'intero più piccolo è $-(2^{31})$ (-2.147.483.648).

Un valore intero può essere espresso in forma decimale, ottale ed esadecimale.

- Per esprimere un valore in forma **ottale** occorre farlo precedere dallo **zero**.
- Per esprimere un valore in forma **esadecimale** occorre farlo precedere da **0x**.

Consideriamo le seguenti inizializzazioni:

```
$InteroDecimalePositivo = 54;  
$InteroDecimaleNegativo = -54;  
$InteroOttale = 042; // Valore 42 espresso in forma ottale  
$InteroEsadecimale = 0x12AF; // Valore 12AF espresso in forma esadecimale
```

Se provassimo a visualizzare i valori di tali variabili nel seguente modo:

```
print($InteroDecimalePositivo); print("<br/>");  
print($InteroDecimaleNegativo); print("<br/>");  
print($InteroOttale); print("<br/>");  
print($InteroEsadecimale);
```

otterremmo il risultato mostrato nella figura a lato. Tutti i valori vengono convertiti in decimale per default.

Per ottenere, invece, le visualizzazioni dei valori nel formato originale ottale ed esadecimale (quello inserito nella fase di inizializzazione), occorre utilizzare le funzioni di conversione **decOct()** e **decHex()**.

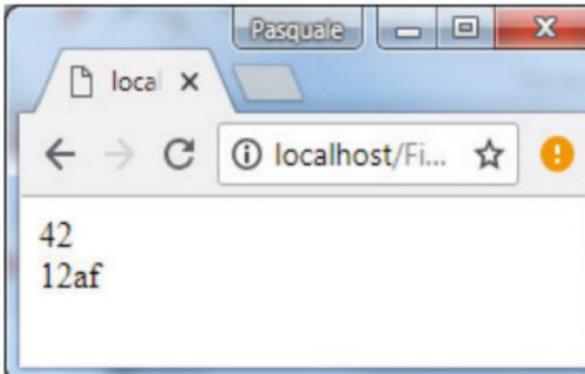
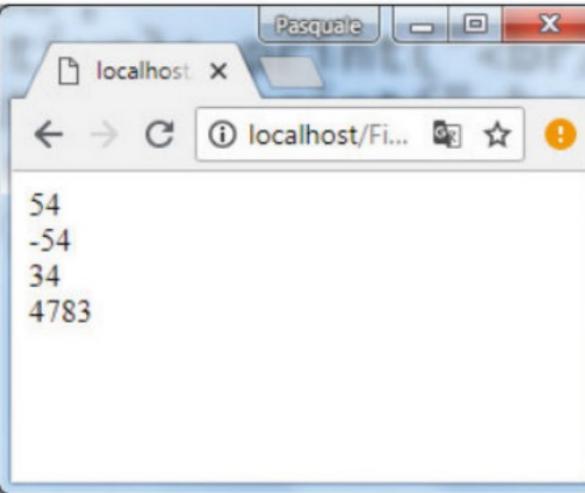
Le istruzioni:

```
print(decOct($InteroOttale));  
print("<br/>");  
print(decHex($InteroEsadecimale));
```

visualizzerebbero quanto mostrato nella figura qui a lato.

Ricorda

Il tag HTML
 si rende necessario per consentire all'interno di una pagina web di andare a capo.



Valori reali

Sono numeri a virgola mobile per i quali si usa la **dot notation anglosassone** e non la virgola. Anche per i reali la dimensione dipende dalla piattaforma, ma generalmente:

- il reale più grande è 1.8E308 con precisione massima di 14 cifre.

Per scrivere un reale si può procedere in uno dei seguenti modi:

```
$Real1 = 5.351;           // Corrisponde a 5,351
$Real2 = -5.351E5;        // Corrisponde a -5,351 * 105, ovvero -535,100
$Real3 = 5E-7;            // Corrisponde a 5 * 10-7, ovvero // 5/10.000.000 = 0,0000005
```

Valori booleani

I valori booleani sono semplicemente TRUE o FALSE (da scrivere in maiuscolo). Ad esempio:

```
$Test = TRUE;  
$B = FALSE;
```

Come vedremo tra poco, PHP considera falso:

- il valore numerico 0, nonché una stringa che contiene ‘0’;
- una stringa vuota;
- un array con zero elementi;
- un valore NULL, cioè una variabile che non è stata definita, o che è stata eliminata con *unset()*, oppure a cui è stato assegnato il valore NULL stesso.

Qualsiasi altro valore per PHP è considerato vero. Quindi qualsiasi numero, intero o decimale purché diverso da 0, qualsiasi stringa non vuota, se usati come espressione condizionale saranno considerati veri.

Valore NULL

Per indicare l'assenza di un valore per una variabile, si utilizza la parola chiave **NULL**. L'istruzione:

```
$Nome = NULL;
```

assegna il valore NULL alla variabile *\$Nome*, ovvero non assegna alcun valore a tale variabile.

Consideriamo il seguente frammento di codice PHP in cui compare un'istruzione condizionale (if) che vedremo tra breve.

```
$V1 = NULL;  
if($V1)  
    print("Sei nel ramo allora");  
else  
    print("Sei nel ramo altrimenti");
```

Se proviamo a eseguire tale codice, il risultato sarà: "Sei nel ramo altrimenti"; la condizione dell'if risulta falsa, in quanto PHP converte in FALSE ogni condizione che restituisce il valore NULL.

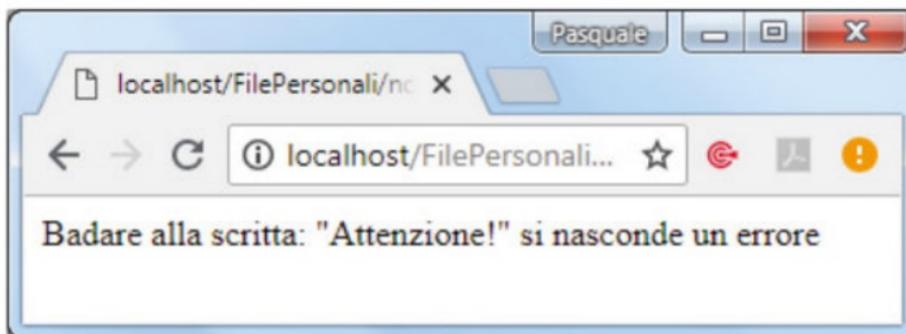
6 Espressioni e valori stringa

Valori stringa

Una stringa è una sequenza di caratteri, senza limitazione, che può essere delimitata da **apici singoli** oppure da **virgolette** (doppie).

Le **stringhe delimitate da apici** sono la forma più semplice, consigliata quando all'interno della stringa non vi sono variabili di cui si vuole ricavare il valore. Ad esempio:

```
$Stringa = ' Badare alla scritta: "Attenzione!" si nasconde un errore ';
print($Stringa);
```



Questo codice visualizzerà quanto mostrato in figura. Verranno visualizzate anche le virgolette ("") che delimitano la parola Attenzione! Quando all'interno di una stringa vi è anche il nome di una variabile

di cui vogliamo visualizzare il valore, ricorriamo alle virgolette. Si dice che, se all'interno di una stringa (**delimitata da virgolette**) esiste il nome di una variabile, questa viene **risolta** da PHP, cioè PHP sostituisce il valore al nome della variabile all'interno della stringa.

Ad esempio, il seguente frammento di codice:

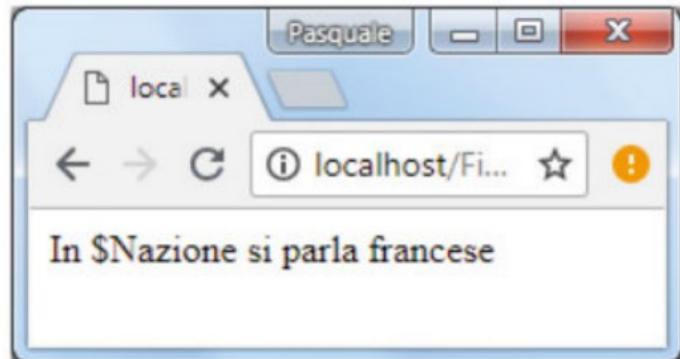
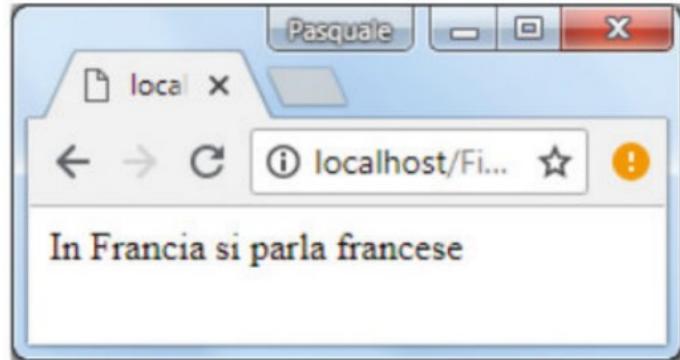
```
$Nazione = 'Francia';
print("In $Nazione si parla francese");
```

visualizzerà quanto mostrato nella figura qui a lato.

Invece:

```
print('In $Nazione si parla francese');
```

visualizzerà quanto mostrato nella figura a destra. L'uso delle virgolette permette di risolvere le variabili, mentre con gli apici singoli i nomi di variabile restano invariati.



Caratteri speciali all'interno delle stringhe

Può capitare che una stringa debba contenere a sua volta apici o virgolette. Per far capire a PHP che quell'apice o quelle virgolette fanno parte della stringa da visualizzare e non sono suoi delimitatori, utilizziamo i caratteri di **escape**, cioè la barra rovesciata (backslash: \) seguita dall'apice o dalle virgolette. Ad esempio, per visualizzare il seguente codice HTML:

```
<a href="Scuola.gif">Istituto Tecnico</a>
```

scrivremo:

```
print("<a href=\"Scuola.gif\">Istituto Tecnico</a>");
```

7 Operatori

Una volta visti i possibili operandi di PHP, introduciamo gli operatori, ultimo tassello per poter manipolare il contenuto delle variabili e in generale per poter comporre espressioni PHP. Gli operatori si dividono in:

- operatori di assegnamento;
- operatori aritmetici binari e unari;
- operatore di concatenamento tra stringhe;
- operatori compattati;
- operatori relazionali;
- operatori logici;
- operatori sui bit;
- un operatore ternario.

Ricorda

I principali operatori che vedrai in questa sezione non dovrebbero risultarti nuovi, in quanto sono gli stessi visti in C/C++.

Operatore di assegnamento

L'operatore di assegnamento o assegnazione, come abbiamo già visto, è il simbolo “=”.

Ad esempio:

```
$Stringa = 'Questa è una nuova stringa';  
$X = 15;  
$Y = $X;
```

Operatori aritmetici

Gli **operatori aritmetici binari**, che si applicano a due operandi, possono essere:

- + addizione;
- - sottrazione;
- * moltiplicazione;
- / quoziente della divisione;
- % modulo (resto della divisione).

Ad esempio:

```
$X = 10 + 20;      // Addizione: X conterrà il valore 30
$Y = $X - 5;      // Sottrazione: Y conterrà il valore 25
$Z = $X * 2;      // Moltiplicazione: Z conterrà il valore 60
$D = $X / 7;      // Divisione: D conterrà il valore 4, cioè il quoziente di 30/7
$R = $X%7;        // Modulo: R conterrà 2, cioè il resto della divisione 30/7
```

Gli operatori **aritmetici unari** possono essere:

- `++` incremento;
- `--` decremento.

Tali operatori si applicano a un solo operando e possono essere posti prima (**prefissi**) o dopo (**postfissi**) rispetto all'operando e il loro comportamento varia secondo questa posizione: l'operatore prefisso modifica l'operando prima di utilizzarne il valore, mentre l'operatore postfisso modifica l'operando dopo averne utilizzato il valore. Ad esempio:

```
$X=10;  
$Y=$X++; // Operatore postfisso: si assegna prima il valore della variabile  
           // $X alla variabile $Y e poi si incrementa la variabile $X  
$X=10;  
$Y=++$X; // Operatore prefisso: prima si incrementa la variabile $X e poi  
           // si assegna il valore alla variabile $Y
```

Operatore di concatenamento tra stringhe

L'operatore di concatenamento o concatenazione rappresentato da “.” è uno degli operatori più utilizzati in PHP, linguaggio nel quale si opera di frequente con le stringhe.

Ad esempio:

```
$Nazione = 'Italia';
$Stringa = 'Nazione di provenienza: ' . $Nazione;
print($Stringa);
```

visualizza la stringa: *Nazione di provenienza: Italia*

Operatori compatti

L'operatore di assegnamento può anche essere **compattato**, cioè **abbinato a un operatore aritmetico**, dando vita a espressioni scritte in forma abbreviata:

$\$X += \Y	equivale a	$\$X = \$X + \$Y$	$\$X /= \Y	equivale a	$\$X = \$X / \$Y$
$\$X -= \Y	equivale a	$\$X = \$X - \$Y$	$\$X %= \Y	equivale a	$\$X = \$X \% \$Y$
$\$X *= \Y	equivale a	$\$X = \$X * \$Y$	$\$X .= \Y	equivale a	$\$X = \$X . \$Y$

Ad esempio:

```
$X = 5 ;
$X += 10;      // Incrementa $X di 10  equivale a $X = $X + 10
$X %= 2;       // Memorizza il valore 1 in $X, equivale a $X = $X % 2
$Str = "Ciao";
$Str .= "a tutti"; // Memorizza "Ciao a tutti" in $Str, equivale a $Str = $Str . "a tutti"
```

Operatori relazionali

Gli operatori relazionali permettono di effettuare un confronto tra gli operandi. Richiedono operandi di tipo primitivo e producono sempre un risultato booleano.

Tutti gli operatori presenti in PHP sono identici a quelli del C++: ==, !=, >, >=, <, <=

In più è presente l'operatore:

- === identico a (si scrive con tre simboli “=” senza spazi tra di loro); il risultato è vero se i due operandi sono uguali tra loro e sono **dello stesso tipo**.

Ad esempio:

```
$X = 5; $Y = 5.0; $Z = 3;          // Assegniamo valori a tre variabili diverse
$X == $Y;                          // Vero hanno lo stesso valore
$X === $Y;                         // Falso, perché $X è intero mentre $Y è reale
$Z >= $X;                          // Falso, $Z è minore di $X
```

Finora abbiamo considerato solo variabili con valori numerici. Gli stessi operatori possono essere applicati anche ad operandi di tipo stringa. In questo caso il confronto viene fatto basandosi sull'ordine alfabetico dei caratteri: vale a dire che vengono considerati *minori* i caratteri che *vengono prima* nell'ordine alfabetico. Quindi "a" è minore di "b", "b" è minore di "c", e così via. Inoltre tutte le lettere minuscole sono *maggiori* delle lettere maiuscole, e tutte le lettere (maiuscole e minuscole) sono *maggiori* delle cifre da 0 a 9. Consideriamo i seguenti esempi:

```
$X = 'Matteo'; $Y = 'Marco'; $Z = 'Giovanni'; $V = 'alberto'; $W = '4  
gatti';  
$X < $Z; // Falso, poiché la lettera 'G' precede la lettera 'M'  
$Y < $X; // Vero, la 'r' ('Mar') precede la 't' ('Mat')  
$Z < $V; // Vero, la 'a' minuscola è 'maggiore' di qualsiasi lettera  
maiuscola  
$Z > $W; // Vero, ogni lettera è 'maggiore' di qualsiasi cifra
```

Operatori logici

Gli operatori logici richiedono come operandi delle espressioni booleane e producono un risultato booleano. Gli operatori logici sono:

Operatore	Simbolo	Significato
or	OR oppure	Or inclusivo: valuta se almeno uno dei due operandi è vero
and	AND oppure &&	And: valuta se entrambi gli operandi sono veri
xor	XOR	Or esclusivo: valuta se uno solo dei due operandi è vero; l'altro deve essere falso
not	!	Negazione: si usa con un solo operando; è vero quando l'operando è falso, e viceversa

Consideriamo i seguenti esempi:

```
10 > 8 AND 7 < 6;          // Falso, perché la prima condizione è vera  
                           // ma la seconda è falsa  
10 > 8 OR 7 < 6;           // Vero  
9 > 5 AND 5 == 5;          // Vero: entrambe le condizioni sono vere  
9 > 5 XOR 5 == 5;          // Falso: solo una delle due deve essere vera perché  
                           // lo 'XOR' sia vero  
4 < 3 || 7 > 9;            // Falso: nessuna delle due condizioni è vera
```

8 Strutture di controllo: i costrutti di selezione

Le istruzioni che controllano il flusso di elaborazione di un programma possono:

- eseguire decisioni condizionate;
- eseguire iterazioni;
- interrompere esecuzioni di blocchi di istruzioni;
- richiamare funzioni.

Anche le strutture di controllo in PHP sono del tutto simili a quelle viste in C/C++; comunque le riporteremo anche in questo contesto, fornendo dei semplici esempi.

Ricordiamo, inoltre, che le espressioni condizionali possono essere tra loro abbinate, facendo attenzione alla precedenza degli operatori.

Il costrutto *if...else*

La più importante istruzione condizionale è il costrutto **if...else**, la cui sintassi è:

```
if(<EspressioneCondizionale>
    <IstruzioniRamoAllora>
[else
    <IstruzioniRamoAltrimenti>]
```

dove:

- <EspressioneCondizionale> può assumere solo i valori TRUE o FALSE. Nel caso in cui l'espressione non abbia un valore booleano, PHP convertirà comunque questo valore in booleano. Le **parentesi tonde** che racchiudono l'espressione condizionale sono **obbligatorie**;
- <IstruzioniRamoAllora> è un blocco di istruzioni che è eseguito quando <EspressioneCondizionale> è vera;
- <IstruzioniRamoAltrimenti> è un blocco di istruzioni che è eseguito quando <EspressioneCondizionale> è falsa.

Esempio: trovare il massimo tra due numeri generati casualmente

```
$N1=rand(0, 10000);           // Restituisce un valore casuale intero
                                // compreso tra 0 e 10000
$N2=rand(0, 10000);
if ($N1 > $N2)
    print("Numero massimo: $N1");
else
    print("Numero massimo: $N2");
```

Nell'esempio abbiamo introdotto la funzione predefinita **rand()** che restituisce un valore intero casuale compreso tra il primo e il secondo parametro impostato da programma.

Abbiamo detto poco fa che l'espressione condizionale potrebbe non avere un valore booleano. PHP però è in grado di considerare booleano qualsiasi valore, in base alle regole viste in precedenza.

Facciamo un altro esempio banale:

```
if (15)
    print("Ciao a tutti!");
```

Questo codice, da un punto di vista logico, non ha nessun senso, ma ci permette di capire come PHP interpreta le nostre espressioni. Infatti in questo caso la stringa “ciao a tutti!” verrà sempre stampata. Questo perché, per PHP, il valore 15, così come qualsiasi numero diverso da 0, è **considerato ‘vero’**.

Il costrutto *if...elseif*

Se occorre effettuare una serie di test si può iterare il ramo *else* in questo modo:

```
if (<EspressioneCondizionale>
    <BloccoIstruzioni>
[elseif (<EspressioneCondizionale>
        <BloccoIstruzioni>]
else
    <BloccoIstruzioni>
```

dove si valuta l'espressione iniziale: se è vera viene eseguito il relativo <BloccoIstruzioni> (dopodiché si termina l'esecuzione), altrimenti si valuta <EspressioneCondizionale> posta accanto a ogni *else* e, se l'espressione è vera, viene eseguito il relativo <BloccoIstruzioni> (in tal caso si termina l'esecuzione), altrimenti si esegue il <BloccoIstruzioni> dell'*else* finale.

Ad esempio vediamo il codice seguente:

```
if ($X == 10 )
    print("Hai inserito un valore di X pari a 10");
elseif($X == 20)
    print("Hai inserito un valore di X pari a 20");
elseif($X == 30)
    print("Hai inserito un valore di X pari a 30");
else
    print("Hai inserito un valore di X diverso da 10, 20 e 30");
```

L'esame dei blocchi *elseif*, e quindi dell'intera *if*, termina appena si incontra una condizione vera; in caso contrario si procede verso l'istruzione *else*.

9 Strutture di controllo: i costrutti iterativi

I **costrutti** o **istruzioni iterative** consentono di ripetere l'esecuzione di un blocco di istruzioni. Analizzeremo le seguenti istruzioni iterative:

- while
- do...while
- for

Il costrutto **while**

Il costrutto di base iterativo precondizionale è **while**, la cui sintassi è:

```
while(<EspressioneCondizionale>
{
    <Istruzioni>
}
```

Viene verificata l'<EspressioneCondizionale> e, se essa è *vera*, viene eseguito il *BloccoIstruzioni*. Appena si esegue l'ultima istruzione del blocco si verifica nuovamente la condizione e, se è nuovamente vera, viene rieseguito il blocco.

Si termina quando l'<Espressione Condizionale> è *falsa*.

Ad esempio, se volessimo leggere 10 numeri da input e stampare la loro somma, ricorreremmo al seguente codice con costrutto iterativo.

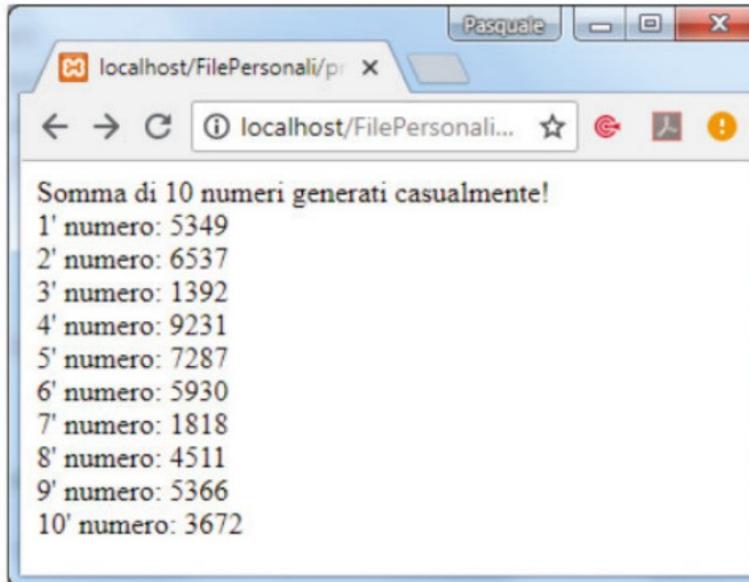
Somma di numeri generati casualmente

Rifletti

Le parentesi tonde che racchiudono l'espressione condizionale sono obbligatorie anche con l'istruzione *while*.

```
<?php
    print("Somma di 10 numeri generati casualmente!<br/>\n");
    $I = 0;
    $Somma = 0;
    while($I<10)
    {
        $Temp= rand(0, 10000); // Genera un numero casuale intero
                               // compreso tra 0 e 10000
        $Somma = $Somma + $Temp;
        $I++;
        print("$I' numero: $Temp <br/>");
    }
    print(" somma = $Somma");
?>
```

Inizialmente la variabile *I*, che funge da variabile contatore, prende il valore 0. Si entra nel ciclo *while*, poiché la condizione è vera (infatti *I* = 0 è minore di 10), si calcola un valore casuale e si incrementa la variabile *I*. Da notare l'utilizzo della funzione *rand(0,10000)*. Il risultato a video sarà quello visibile in figura.



Il costrutto **do...while**

Se occorre eseguire il blocco di istruzioni almeno una volta, ci viene in aiuto un altro controllo di flusso, il costrutto postcondizionale **do...while**, avente la seguente sintassi:

```
do
{
    <BloccoIstruzioni>
}
while(<EspressioneCondizionale>);
```

che esegue il blocco di istruzioni finché l'<EspressioneCondizionale> è vera. A differenza del costrutto *while* visto precedentemente, <BloccoIstruzioni> viene eseguito *almeno una volta*.

Questa è la caratteristica peculiare del ciclo *do...while*: il controllo della condizione avviene solo dopo la prima iterazione. Questo permette di valutare delle *condizioni* che fanno uso di variabili generate all'interno della prima iterazione.

Rifletti

A differenza del C/C++ o di altri linguaggi l'utilizzo di funzioni matematiche non richiede l'inclusione di alcuna libreria all'interno dello script.

Vediamo un esempio, in cui a ogni iterazione viene generato un numero casuale compreso tra 0 e 10. Se il numero generato è più piccolo di 8 il ciclo continua, altrimenti viene interrotto.

```
<?php
do
{
    $Numero = rand(0,10);          // Genera un numero casuale compreso
                                    // tra 0 e 10
    print($Numero . "<br/>\n"); // Mostra il numero e va a capo
}
while($Numero < 8);           // Se $Numero < 8 ripete le istruzioni
?>
```

Anche il ciclo *do...while*, come il ciclo *while*, è indicato quando il numero di iterazioni non è noto a priori (non sappiamo quando verrà generato un numero uguale o superiore a 8). Da notare che nell'esempio precedente è necessario generare il primo numero casuale per decidere se andare avanti con le iterazioni oppure no. È quindi necessario eseguire almeno una volta il ciclo per poter valutare la condizione sulle iterazioni successive.

Il costrutto *for*

Un altro costrutto iterativo molto usato è il costrutto iterativo **for**.

La sua sintassi è:

```
for (<EspressioneIniziale>; <EspressioneCondizionale>; <Passo>)
{
    <BloccoIstruzioni>
}
```

dove *<EspressioneIniziale>* e *<Passo>* sono due espressioni; generalmente la seconda consiste nell' incremento di una variabile.

L'esecuzione del costrutto *for* avviene nel seguente modo: si esegue l'*<EspressioneIniziale>* e si verifica l'*<EspressioneCondizionale>*. Se quest'ultima è *vera* si esegue il blocco di istruzioni. A ogni ciclo si esegue *<Passo>* e si verifica di nuovo l'*<EspressioneCondizionale>*. Quando quest'ultima diviene falsa, si termina l'esecuzione.

Ad esempio, per sommare i primi 100 numeri naturali scriveremo:

```
for ($I=1; $I <= 100; $I++)
    $Somma=$Somma+$I;
```

In questo esempio la variabile *\$I* viene chiamata *variabile contatore*.

Il blocco di istruzioni è composto da un'unica istruzione, quindi *non sono presenti le parentesi graffe*.

OSSERVA COME SI FA

- Confrontare i tre costrutti iterativi

Creare una tabella html contenente Quantità e Prezzo degli articoli presenti in magazzino

Creiamo tre differenti programmi PHP per risolvere il problema. I tre programmi producono lo stesso risultato finale e sono relativi all'utilizzo dei tre costrutti iterativi: *while*, *do...while* e *for*.

Analizziamo il codice con il costrutto *while*.

```
<?php
$PrezzoBase = 5;
$Contatore = 10;

echo "<table style=\"border: 1px solid black; text-align:center;\">>";
echo "<tr><th>Quantita'</th>";
echo "<th>Prezzo</th></tr>";
while($Contatore <= 100)
{
    echo "<tr><td>";
    echo $Contatore;
    echo "</td><td>";
    echo $PrezzoBase * $Contatore;
    echo "</td></tr>";
    $Contatore = $Contatore + 10;
}
echo "</table>";
?>
```

Analizziamo il codice con il costrutto *do...while*.

```
<?php
$PrezzoBase = 5;
$Contatore = 10;
echo "<table style=\"border: 1px solid black; text-align:center;\">";
echo "<tr><th>Quantita'</th>";
echo "<th>Prezzo</th></tr>";
do {
    echo "<tr><td>";
    echo $Contatore;
    echo "</td><td>";
    echo $PrezzoBase * $Contatore;
    echo "</td></tr>";
    $Contatore = $Contatore + 10;
} while($Contatore <= 100);
echo "</table>";
?>
```

Analizziamo il codice con il costrutto *for*.

```
<?php
$PrezzoBase = 5;

echo "<table style=\"border: 1px solid black; text-align:center;\">";
echo "<tr><th>Quantita'</th>";
echo "<th>Prezzo</th></tr>";
for($Contatore = 10; $Contatore <= 100; $Contatore += 10)
{
    echo "<tr><td>";
    echo $Contatore;
    echo "</td><td>";
    echo $PrezzoBase * $Contatore;
    echo "</td></tr>";
}
echo "</table>";
?>
```

Il ciclo crea una nuova tabella, con due colonne (una per la *Quantità*, l'altra per il *Prezzo*) finché la variabile *Contatore* non arriva al valore di 100. Quando il valore della variabile *Contatore* supera 100, la condizione fallisce e il ciclo termina. Vediamo in dettaglio che cosa succede:

- vengono impostate le due variabili *\$PrezzoBase* e *\$Contatore* ai valori iniziali;
- vengono impostati i *tag* iniziali della tabella html e dei suoi *header*;
- viene verificata la condizione del *while*, che è minore di 100, poiché *\$Contatore* è pari a 10 (valore iniziale);
- viene eseguito il codice all'interno del *while*, creando una nuova riga per il prezzo pari a 10;
- viene aggiunto 10 alla variabile *\$Contatore* per portare il suo valore a 20;
- il ciclo riparte dal passo 3, finché la variabile *\$Contatore* sarà maggiore di 100;
- subito dopo il ciclo vengono inseriti i *tag* di chiusura della tabella.

Da notare gli *slash* posti prima dei doppi apici nella prima istruzione *echo* (quelli a sinistra di *border*). Se questi fossero omessi, i secondi doppi apici (quelli posti vicino a *border*) sarebbero interpretati come la chiusura della stringa che l'istruzione *echo* deve visualizzare.

10 Definire un array

Array monodimensionali

In PHP un **array** o **vettore** monodimensionale è un modo di raggruppare un insieme di valori diversi e di associarli ad appropriate chiavi di accesso (indici). Tali chiavi sono generalmente di tipo numerico, ma possono anche essere di tipo alfanumerico: nel secondo caso si parla di **array associativi**. A differenza di molti linguaggi di programmazione, in PHP i valori contenuti in un array possono essere di diverso tipo. Possono, cioè, esistere array di dati non omogenei. Anche la dimensione degli array può variare nel corso dell'esecuzione di uno script.

È possibile definire un array monodimensionale in diversi modi.

- Un primo modo è quello di elencare gli elementi uno a uno (l'indice inizia da 0). Ad esempio:

```
$Animali[0] = 'Anatra';
$Animali[1] = 'Orso';
$Animali[2] = 'Gallina';
$Animali[3] = 'Asino';
$Animali[4] = 'Rana';
```

Ricorda

L'indice del vettore inizia da 0; pertanto in un vettore di n elementi l'ultimo elemento avrà come indice la posizione n-1.

- Un secondo modo è quello di elencare gli elementi separandoli con virgole, utilizzando la funzione predefinita *array()*. In questo modo si definisce e si **inizializza** un array.

Ad esempio:

```
$Animali = array('Anatra', 'Orso', 'Gallina', 'Asino', 'Rana');
```

Usando questo tipo di definizione, PHP associa a ciascuno dei valori che abbiamo elencato un indice numerico, a partire da 0. Quindi, in questo caso, ‘Anatra’ assumerà l’indice 0, ‘Orso’ l’indice 1, e così via fino a ‘Rana’ che avrà indice 4.

Per **riferirsi a un singolo elemento** dell’array si indica il nome dell’array seguito dall’indice racchiuso fra parentesi quadre:

```
print($Animali[1]); // Visualizza 'Orso'
```

Per creare l’array *\$Animali* vuoto, è possibile scrivere:

```
$Animali = array( );
```

- Un terzo modo può essere usato anche come metodo per **aggiungere** un valore all'array (istruzione di **incremento** o **inserimento**):

```
$Animali[ ]= 'Leone';
```

Con questa sintassi viene creato un nuovo elemento nell'array *\$Animali*. Se l'array *\$Animali* conteneva 5 elementi (con indice da 0 a 4), il nuovo elemento aggiunto avrebbe indice 5, verrebbe cioè “aggiunto un elemento in fondo all'array”.

Come abbiamo detto, questa sintassi è valida anche per definire un array, in quanto se l'array *\$Animali* non fosse ancora stato definito, questa istruzione lo avrebbe definito, creando l'elemento con indice 0.

È naturalmente possibile indicare direttamente l'indice; ad esempio:

```
$Animali[3] = 'Bue';
$Animali[7] = 'Cavallo';
```

Dopo queste istruzioni, l'elemento con indice 3, che prima valeva ‘Asino’, avrà il valore ‘Bue’. Inoltre avremo un nuovo elemento di indice 7, con il valore ‘Cavallo’.

È da notare che, dopo queste istruzioni, il nostro array ha un “buco”, perché dall'indice 5 si salta direttamente all'indice 7: successivamente, se usassimo di nuovo l'istruzione di “incremento” con le parentesi quadre vuote, il nuovo elemento prenderà l'indice 8.

Infatti PHP, con un'istruzione di quel tipo, va a cercare l'elemento con l'indice più alto e lo aumenta di 1 per creare quello nuovo.

Array associativi

Gli indici degli elementi di un array non sono necessariamente numerici, ma possono essere anche delle stringhe: in questo caso si parla di **array associativi**.

Ad esempio:

```
$AnimaliAfricani['nome'] = 'Giraffa';
```

Con questa istruzione abbiamo definito un array di nome *\$AnimaliAfricani*, creando un elemento la cui chiave (indice) è ‘nome’ e il cui valore è ‘Giraffa’.

Chiavi o indici numerici e associativi possono coesistere nello stesso array.
Consideriamo il seguente esempio relativo al personale coinvolto nel team di sviluppo di una nuova applicazione software:

```
$TeamDiSviluppo[1] = 'Mario';
$TeamDiSviluppo[2] = 'Carlo';
$TeamDiSviluppo[3] = 'Fabio';
$TeamDiSviluppo[4] = 'Paolo';
$TeamDiSviluppo[5] = 'Luigi';
$TeamDiSviluppo[6] = 'Marco';
$TeamDiSviluppo['analista'] = 'Ettore';
```

In questo caso abbiamo creato un array con sette elementi, di cui sei con chiavi numeriche e uno con chiave associativa. Se in seguito volessimo aggiungere un elemento usando le parentesi quadre vuote, il nuovo elemento prenderà l'indice 7.

Avremmo potuto creare lo stesso array usando l’istruzione di dichiarazione dell’array, nel seguente modo:

```
$TeamDiSviluppo = array(1 => 'Mario', 'Carlo', 'Fabio', 'Paolo',
                        'Luigi', 'Marco', 'analista' => 'Ettore');
```

Analizziamo il formato di quest’ultima istruzione: per prima cosa abbiamo creato il primo elemento, assegnandogli esplicitamente la chiave 1 invece che la chiave 0 che, come sappiamo, il sistema avrebbe assegnato di default.

Per fare questo abbiamo indicato la chiave, seguita dal simbolo => (uguale seguito da maggiore) e dal valore dell’elemento.

Per gli elementi successivi ci siamo limitati a elencare i valori, in quanto PHP, per ciascuno di essi, crea la chiave numerica aumentando di 1 la più alta già esistente. Quindi ‘Carlo’ prende l’indice 2, ‘Fabio’ il 3, e così via.

Arrivati all’ultimo elemento, siccome vogliamo assegnargli una chiave associativa, siamo obbligati a indicarla esplicitamente.

Quando abbiamo usato le chiavi associative, le abbiamo indicate fra apici: ciò è necessario per mantenere la “pulizia” del codice, in quanto, se non usassimo gli apici (come spesso viene fatto), PHP genererebbe un errore di tipo *notice*, anche se lo script funzionerebbe ugualmente.

L'eccezione si ha quando l'elemento di array viene indicato fra virgolette:

```
$TeamDiSviluppo['analista'] = 'Ettore';      // Corretto
$TeamDiSviluppo[progettista] = 'Piero';       // Funziona, ma genera un errore
                                                // di tipo 'notice'
print $TeamDiSviluppo['progettista']; // Corretto: visualizza 'Piero'
print "signor $TeamDiSviluppo[analista]";
    /* Corretto senza apici fra le virgolette: visualizza 'signor Ettore'*/
print "signor $TeamDiSviluppo['analista']"; // Attenzione: non funziona,
                                                // genera errore
print "signor {$TeamDiSviluppo['analista']}";
    /* Corretto: per usare gli apici fra virgolette dobbiamo comprendere tutto
fra parentesi graffe */
print "signor " . $TeamDiSviluppo['analista'];/* Corretto: come alternativa,
usiamo il punto per concatenare*/
```

Quando un elemento dell'array è inserito all'interno di una stringa, le chiavi associative devono essere riportate senza apici.

OSSERVA COME SI FA

1. Invertire un array

Creiamo un programma PHP che visualizzi un vettore di numeri interi nell'ordine inverso rispetto a quello in cui è assegnato.

Analizziamo due situazioni diverse: assumiamo in una che l'array `$C` contenga `$N` valori e che `$N` sia assegnato in ingresso. Assumiamo, poi, che il numero di elementi dell'array `$D` non sia noto a priori, ma che il vettore contenga numeri positivi e sia "terminato" dal valore `-1`.

```
<?php
    $N = 10;
    $C = array( 10, 3, 1, 100, 230, 56, 33, 53, 91, 24 );
    print "<p>Il primo vettore contiene, in ordine inverso: ";
    for ( $I = $N - 1; $I >= 0; $I-- )
        print $C[ $I ] . " ";
    print "</p>";
    // $D contenga numeri positivi e sia "terminato" dal valore -1
    $D = array( 10, 350, 91, 24, -1 );
    $N = 0;
    for ($I = 0; $D[$I] > 0; $I++) // Contiamo gli elementi di $D
        $N = $N + 1;
    print "<p>Il secondo vettore contiene, in ordine inverso: ";
    for ($I = $N - 1; $I >= 0; $I--)
        print $D[ $I ] . " ";
    print "</p>";
?>
```

2. Invertire un array formattando il risultato in una tabella HTML

Consideriamo l'esempio precedente e questa volta formattiamo il risultato in una tabella le cui celle hanno un colore casuale.

Il codice è:

```
<?php
$N = 10;
$D = array( 10, 3, 1, 100, 230, 56, 33, 53, 91, 24 );
print "<p>Il primo vettore contiene, in ordine inverso: </p>";
print("<table style=\"border: 1px solid black; \"><tr>");
for ($I = $N - 1; $I >= 0; $I--) // $R, $G, $B rappresentano le componenti
                                // di colore per il codice RGB
{
    $R=rand(0,255);
    $G=rand(0,255);
    $B=rand(0,255);
    print"<td style=\"background-color:rgb($R,$G,$B)\">>". $D[$I] . "</td>";
}
print("</tr></table>");
$D = array(10, 350, 91, 24, -1);
$N = 0;

for($I = 0; $D[$I] > 0; $I++)
{
    $N = $N + 1;
print "<p>Il secondo vettore contiene, in ordine inverso: </p>";
print "<table style=\"border: 2px solid black; \"><tr>";
for($I = $N - 1; $I >= 0; $I--)
{
    $R=rand(0,255);
    $G=rand(0,255);
    $B=rand(0,255);
    print "<td style=\"background-color:rgb($R,$G,$B)\">>". $D[ $I ] . "</td> ";
}
print "</tr></table>";
?>
```

Otterremo quanto è mostrato nella figura.

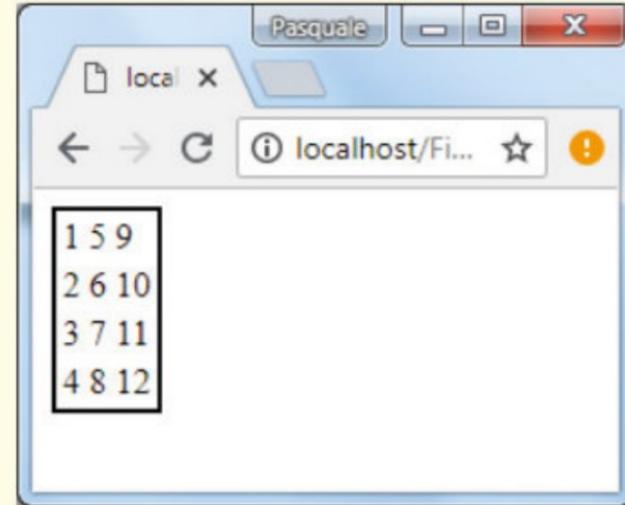


3. Visualizzare la trasposta di una matrice

Data una matrice $M(j,i)$, la sua trasposta M' , ossia la matrice ottenuta scambiando le righe con le colonne, è ottenuta come $M'(i,j) = M(j,i)$.

Nell'esempio che segue inizializziamo con valori interi la matrice M e visualizziamo la sua trasposta M' usando una tabella html.

```
<?php
$C = 4;
$R = 3;
$D = array(
    array(1, 2, 3, 4),
    array(5, 6, 7, 8),
    array(9,10, 11, 12)
);
print"<table style=\"border: 2px solid black; \">>";
for ($I = 0; $I < $C; $I++)
{
    print"<tr>";
    for($J = 0; $J < $R; $J++)
        print"<td>". $D[$J][$I]. "</td>";
    print"</tr>";
}
print "</table>";
?>
```



Una matrice è un array bidimensionale in cui ogni riga è costituita da un singolo array monodimensionale.

11 Il costrutto *foreach*

foreach per array monodimensionali

I costrutti iterativi (*for*, *while* e così via), benché molto potenti e versatili, possono risultare piuttosto macchinosi quando è necessario scorrere gli elementi di un array. In questi casi PHP mette a disposizione un comodo costrutto: il ciclo *foreach*, che ha la seguente sintassi:

```
foreach(<ArrayDaScorrere> as <ValoreElemento>)
{
    <Istruzioni>      // Istruzioni da eseguire per ogni elemento dell'array
}
```

Le istruzioni raggruppate nelle parentesi graffe verranno ripetute tante volte quanti sono gli elementi presenti nell'array <ArrayDaScorrere>. A ogni iterazione, un valore presente in tale array verrà letto e copiato nella variabile <ValoreElemento>.

Vediamo un esempio:

```
<?php
$Nazioni = array('italia','francia','germania','spagna','danimarca','ungheria');
foreach($Nazioni as $Stato)
{
    print($Stato . '<br/>');
}
?>
```

A ogni iterazione la variabile `$Stato` assumerà il valore di un elemento presente nell'array `$Nazioni`. Un ciclo *foreach* serve a iterare delle operazioni che vanno eseguite sugli elementi di un array.

Consideriamo l'esempio precedente e vediamo che cosa succede quando usiamo il ciclo *foreach* per effettuare una modifica ai valori contenuti nell'array `$Nazioni`.

```
<?php
$Nazioni =
array('italia','francia','germania','spagna','danimarca','ungheria');
foreach($Nazioni as $Stato)
{
    $Stato = strtoupper($Stato);
}
foreach($Nazioni as $UnoStato)
{
    print($UnoStato. '<br/>');
}
?>
```

Abbiamo percorso 2 volte l'array `$Nazioni`. Una prima volta per modificare ogni suo elemento in modo che tutti i nomi delle nazioni risultassero maiuscoli, utilizzando una funzione predefinita `strtoupper()`. La seconda volta, invece, per mostrarne il contenuto, che però risulta inalterato. Tale comportamento è riconducibile al fatto che la variabile `$Stato` contiene solo la copia di un valore dell'array `$Nazioni` e pertanto non è direttamente collegata a esso. La modifica alla variabile `$Stato`, quindi, non influisce sul relativo valore dell'array `$Nazioni`. Per risolvere questo problema ed effettuare le modifiche ai valori dell'array basterà utilizzare l'operatore `&` (operatore di reference). Occorrerà porre il carattere `&` prima della variabile `$Stato` nell'espressione del ciclo `foreach` per far sì che ogni modifica effettuata su di essa venga in realtà effettuata sul relativo valore dell'array. Riscriviamo l'esempio in modo identico al precedente, tranne per il carattere `&` posto a sinistra della variabile `$Stato`:

```
<?php
$Nazioni = array('italia','francia','germania','spagna','danimarca','ungheria');
foreach($Nazioni as  &$Stato)
{
    $Stato = strtoupper($Stato);
}
foreach($Nazioni as $UnoStato)
{
    print($UnoStato . '<br/>');
}
?>
```

***foreach* per array associativi**

Il costrutto *foreach* può essere utilizzato anche con gli array associativi. È possibile, cioè, scorrere un array e recuperare i suoi elementi sotto forma di coppie chiave => valore. La sintassi è:

```
foreach (<ArrayDaScorrere> as <ChiaveElemento> => <ValoreElemento>)
{
    <Istruzioni>      // Istruzioni da eseguire per ogni elemento dell'array
}
```

Rispetto alla sintassi utilizzata finora, adesso anche le chiavi di ciascun elemento vengono lette e copiate nella variabile <ChiaveElemento>. Vediamo un semplice esempio:

```
<?php
$Nazioni = array('nome' => 'francia', 'capitale' => 'parigi' ,
                  'popolazione' => '70 milioni');
foreach($Nazioni as $Info => $Valore)
{
    print("$Info: $Valore <br/>\n");
}
?>
```

Anche con questa sintassi è possibile usare il carattere & per far sì che la variabile \$Valore sia legata direttamente al corrispondente valore dell'array da scorrere, in modo che questo possa essere modificato.

***foreach* per array multidimensionali**

Il costrutto *foreach* viene utilizzato anche per gli array multidimensionali. Facciamo subito un esempio in cui visualizziamo i dati relativi a un array multidimensionale di nazioni. Si tratta di un array i cui elementi sono, a loro volta, altri array contenenti i dati relativi al nome della nazione, alla sua capitale e alla sua popolazione.

```
<?php
    $Nazioni = array(array('nome'=>'francia', 'capitale'=>'parigi',
        'popolazione'=>'75 milioni'),
        array('nome' => 'italia', 'capitale' => 'roma' , 'popolazione'=>'60
        milioni'));
    foreach($Nazioni as $Info => $Valore)
    {
        print('Nazione: ' . $Valore['nome']. '<br/>');
        print('Capitale: ' . $Valore['capitale']. '<br/>');
        print('Popolazione: ' . $Valore['popolazione']. '<br/>');
        print('<br/><br/>');
    }
?>
```

Notiamo come la variabile *\$Valore* sia ancora un array e pertanto i dati in essa contenuti siano accessibili attraverso l'uso delle diverse chiavi (nazione, capitale, popolazione). Tale approccio è comunemente usato anche nell'estrazione di dati da un database. Questi, infatti, spesso sono disponibili per un programma PHP proprio sotto forma di array

12 Funzioni definite dall'utente

Come ogni linguaggio di scripting che si rispetti, anche PHP ha la possibilità di effettuare **astrazioni procedurali**, cioè consente di assegnare un nome a porzioni di codice in modo da poterle riutilizzare richiamandole. A questo scopo, PHP utilizza le funzioni.

Una **funzione** è un insieme di istruzioni che hanno lo scopo di eseguire determinate operazioni.

La convenienza di ricorrere alle funzioni sta nel fatto che non occorre riscrivere tutto il codice ogni volta che abbiamo la necessità di eseguire quelle operazioni: basta infatti richiamare l'apposita funzione, fornendole i parametri, cioè i dati, di cui ha bisogno per la sua esecuzione. In PHP vi sono due tipi di funzioni:

- funzioni **incorporate** nel linguaggio o **predefinite**;
- funzioni **definite dall'utente**.

In precedenza abbiamo già avuto modo di utilizzare le funzioni predefinite; soffermiamoci ora su quelle definite dall'utente. Sia le funzioni predefinite, sia quelle definite dall'utente si richiamano allo stesso modo. Ciò che caratterizza le funzioni, oltre ovviamente ai compiti che svolgono, è il fatto di richiedere **parametri di ingresso** e fornire **valori di ritorno**. Come abbiamo già visto per le funzioni predefinite, la sintassi fondamentale con la quale si richiama una funzione, cioè si chiede a PHP di eseguirla, consiste nell'indicarne il nome e nell'elencare tra parentesi tonde gli eventuali parametri attuali:

```
<NomeFunzione>([<ParametriAttuali>]);
```

Le parentesi tonde devono obbligatoriamente essere indicate anche se non ci sono parametri attuali. Questi ultimi, se presenti, sono separati da virgola.

Nel caso in cui la funzione restituisca un valore, possiamo indicare la variabile (o l'array) che utilizzerà questo valore:

```
<NomeVariabile> = <NomeFunzione>([<ParametriAttuali>]);
```

Definizione di funzioni

Per definire una funzione utilizziamo la parola chiave **function**, seguita dal nome che abbiamo individuato per la funzione e dalle parentesi che contengono gli eventuali parametri formali (nomi di variabili preceduti da \$); a seguire, fra parentesi graffe, ci sarà il codice che viene eseguito ogni volta che la funzione viene richiamata.

```
function <NomeFunzione>(<Parametro1>, ..., <ParametroN>)
{
    <Istruzioni>
}
```

Ad esempio, consideriamo una funzione che, dati due numeri, restituisca il maggiore dei due:

```
function Maggiore($N1, $N2)
{
    if ($N1 > $N2)
        return $N1;
    else
        return $N2;
}
```

All'interno della funzione notiamo l'istruzione *return*, che segna il termine della funzione stessa, restituendo il controllo allo script chiamante nel punto in cui la funzione è stata chiamata e contemporaneamente restituendo l'eventuale valore di ritorno.

Nell'esempio precedente, i due dati ricevuti in input vengono controllati e la funzione termina immediatamente, restituendo il valore maggiore tra i due.

Per richiamare la funzione possiamo scrivere il frammento di codice che segue, in un punto qualsiasi dello script principale che la utilizza, poiché l'**ambito dei nomi di funzioni è globale**, cioè una funzione definita in uno script è disponibile dovunque all'interno di esso.

```
$X = 10;  
$Y = 24;  
$M = Maggiore($X, $Y);           // Il contenuto di $M sarà 24
```

È importante ricordare che non c'è alcuna relazione definita tra i nomi dei parametri formali che la definizione della funzione utilizza e quelli dei parametri attuali (indicati al momento della chiamata). Ciò che determina la corrispondenza fra gli argomenti è, infatti, semplicemente la posizione in cui essi vengono indicati: nel nostro caso, quindi, \$X diventerà \$N1 all'interno della funzione e \$Y diventerà \$N2.

Come possiamo osservare dalla precedente sintassi di definizione di una funzione, i tipi dei parametri formali e il tipo del valore di ritorno non devono essere dichiarati.

Visibilità delle variabili locali e globali

Le variabili e i nomi dei parametri formali utilizzati in una funzione esistono solo all'interno della funzione stessa, mentre non sono definiti né per le altre funzioni, né nello script principale. Allo stesso modo, le variabili usate dallo script principale non vengono viste dalle funzioni.

Se, ad esempio, avessimo scritto `print($N2)` all'esterno della funzione, non avremmo ottenuto alcun risultato, e anzi avremmo ricevuto un errore di tipo *notice*, in quanto la variabile `$N2`, in quell'ambito, non è definita.

Le variabili utilizzate all'interno di una funzione si chiamano **variabili locali**. Le variabili utilizzate dallo script principale, invece, sono dette **variabili globali**.

Rifletti

È buona norma di programmazione fare ricorso alle **variabili globali** solo quando è indispensabile.

Per fare riferimento a una variabile globale all'interno di una funzione, occorre dichiararla attraverso l'istruzione **global**. Ad esempio:

```
function Maggiore($N1, $N2)
{
    global $T;
    print("Verifica $T maggiore");
    if ($N1 > $N2)
        return $N1;
    else
        return $N2;
}
$T = 'Temperatura';
$T1=12;
$T2=23;
$M=Maggiore($T1, $T2);
```

Come possiamo notare dall'esempio, se non ci fosse l'istruzione *global \$T*, la variabile *\$T* risulterebbe non definita all'interno della funzione, e quindi l'istruzione *print* genererebbe un errore.

13 Passaggio di dati tra HTML e PHP

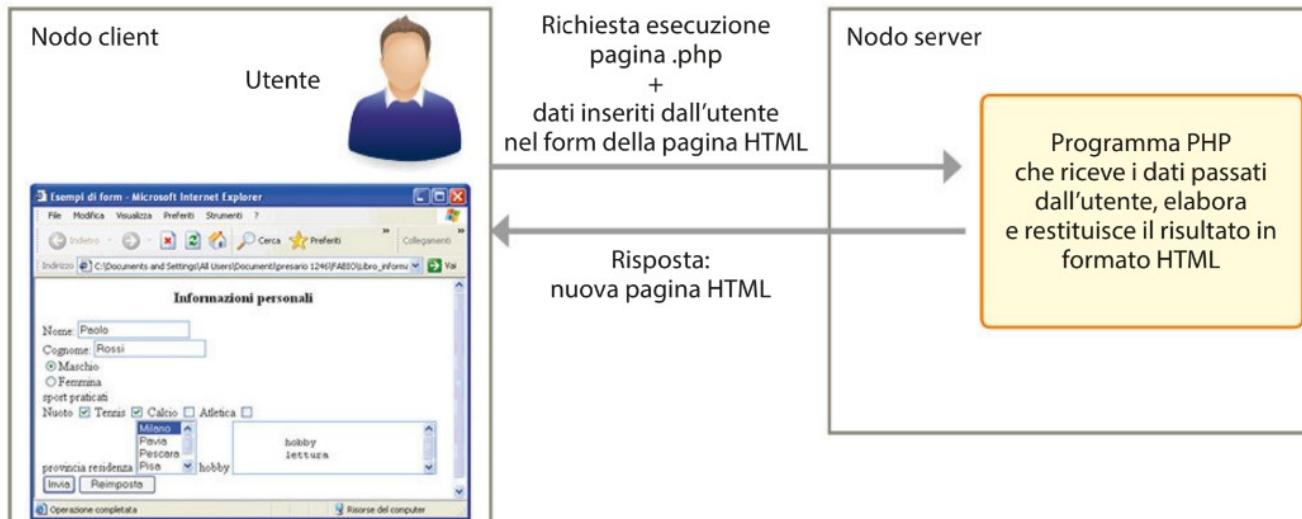
Finora ci siamo occupati delle caratteristiche di base del linguaggio PHP e ci siamo occupati di come l'utente può richiamare un programma; non abbiamo ancora analizzato come l'utente possa interagire con un programma PHP passandogli, ad esempio, i dati necessari all'elaborazione.

Negli esempi fin qui svolti, infatti, i dati iniziali per le nostre funzioni o per le nostre variabili non sono mai stati inseriti dall'utente.

Uno strumento molto utile per passare le informazioni utente al server è il form, al cui interno si possono utilizzare diversi tipi di elementi. Tale passaggio avviene utilizzando due differenti metodi:

- il metodo GET;
- il metodo POST.

Entrambi riguardano l'interazione tra un programma PHP che risiede su un server web e un documento HTML che è stato scaricato sul computer dell'utente.



14 Il metodo GET

Il **metodo GET** consente di accodare i dati all'indirizzo della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e da coppie NomeParametro/ValoreParametro con i dati che ci interessano.

NomeParametro e ValoreParametro sono separati da un segno di uguale =. Le diverse coppie nome/valore sono separate dal segno &. La sintassi è:

```
http://UrlDelServer/Cammino/Programma.php?NomeParametro1=
    ValoreParametro1&NomeParametro2=ValoreParametro2&.....
    &NomeParametroN=ValoreParametroN
```

La stringa che si trova dopo il punto interrogativo, contenente nomi e valori dei parametri, è detta **query string**. Quando la pagina Programma.php viene richiamata in questo modo, potrà fare riferimento ai parametri ricevuti utilizzando la seguente sintassi:

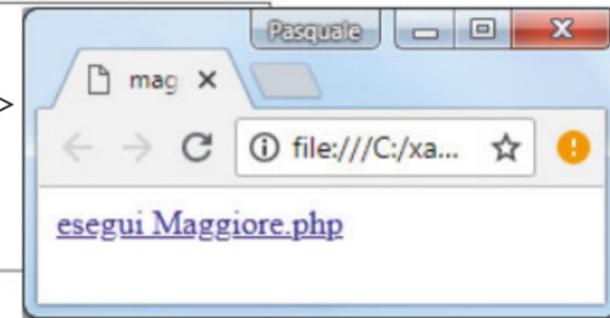
```
$_GET['NomeParametro']
```

I valori contenuti nella query string vengono memorizzati da PHP nell'array **\$_GET**, che è un array **superglobale**, in quanto è disponibile anche all'interno delle funzioni.

Consideriamo l'esempio del maggiore tra due numeri visto in precedenza e modifichiamolo nel modo mostrato in figura: l'esecuzione del file MaggioreGet.php avverrà attraverso il link posto nel file Maggiore.html. Tale link rimanda a un URL contenente i valori da passare alla funzione.

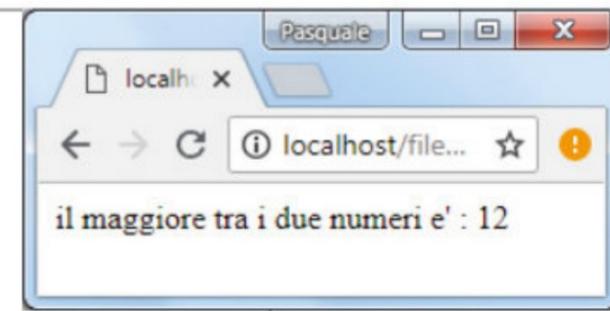
File Maggiore.html

```
<html>
  <body>
    <a href="http://localhost/MaggioreGet.php?Num1=2&Num2=12">
      esegui Maggiore.php
    </a>
  </body>
</html>
```



File MaggioreGet.php

```
<?php
  function Maggiore($N1, $N2)
  {
    if ($N1 > $N2)
      return $N1;
    else
      return $N2;
  }
  $M = Maggiore($_GET['Num1'], $_GET['Num2']);
  print("il maggiore tra i due numeri e' : $M ");
?>
```



Passaggio di parametri da HTML a PHP con il metodo GET

Un miglioramento di quest'esempio consiste nel permettere all'utente l'inserimento dei valori dei due numeri tramite **campi di input**; in questo modo è possibile riutilizzare il programma per qualsiasi valore dei dati in input. Il file PHP rimane invariato, cambia solo il file HTML, che diventa:

File Maggiore2.html

```
<html>
  <body>
    <form action="http://localhost/FilePersonali/MaggioreGet.php"
      method="GET">
      <input type="Text" name="Num1" /> Inserisci il primo numero <br/>
      <input type="Text" name="Num2" />Inserisci il secondo numero <br/>
      <input type="Submit" value="esegui Maggiore.php" /> <br/>
    </form>
  </body>
</html>
```

Il risultato visualizzato sullo schermo è mostrato in figura. Da notare che:

- abbiamo inserito la stringa contenente l'URL del server e il nome del programma PHP nella proprietà **action** di un form HTML;
- abbiamo usato l'altra proprietà **method** per specificare il metodo che stiamo usando, per l'appunto GET;
- abbiamo usato la proprietà **name** per dare il nome ai parametri dei campi **input** Num1 e Num2;
- abbiamo utilizzato l'elemento **SUBMIT**, il quale visualizza un pulsante ed è preposto all'invio dei dati dal form verso il file indicato nella proprietà *action*.

The screenshot shows a web browser window titled "Pasquale". The address bar displays "file:///C:/xampp/magic". The main content area contains a form with two input fields and a submit button. The first field is labeled "Inserisci il primo numero" and the second is labeled "Inserisci il secondo numero". Below the fields is a button labeled "esegui Maggiore.php".

Il metodo GET, nonostante la sua semplicità, presenta alcuni indubbi svantaggi:

- non è adatto per lo scambio di dati relativi ai login, poiché sia il nome utente sia la password risulterebbero completamente visibili a video e memorizzabili dal browser del client come pagina visitata;
- ogni invio con il metodo GET viene registrato nel file di registro del web server, compresi i valori dei parametri passati nella query string;
- la lunghezza dell'URL è limitata, poiché tale URL viene assegnato a una variabile del server.

Queste considerazioni ci portano a preferire il metodo POST, descritto nel paragrafo seguente, almeno quando vi è la necessità di passare dati inseriti nei form HTML.

15 Il metodo POST

Il **metodo POST** viene utilizzato con i form specificando nella proprietà **METHOD** il valore POST. I dati che il browser invia al server attraverso la richiesta HTTP non sono visibili sulla barra degli indirizzi.

I dati passati attraverso il metodo POST sono memorizzati nell'array **`$_POST`** che, proprio come **`$_GET`**, è un array **superglobale**.

Riprendiamo l'ultimo esempio e modifichiamolo in modo da utilizzare il metodo POST per il passaggio dei parametri.

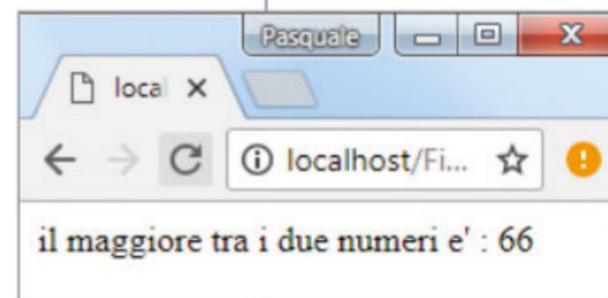
Utilizzo del metodo POST con i FORM

File Maggiore3.html

```
<html>
  <body>
    <form action="http://localhost/FilePersonali/MaggiorePost.php"
      method="POST">
      <input type="Text" name="Num1" />Inserisci il primo numero <br/>
      <input type="Text" name="Num2" />Inserisci il secondo
      numero <br/>
      <input type="Submit" value="esegui Maggiore.php" /> <br/>
    </form>
  </body>
</html>
```

File MaggiorePost.php

```
<?php
function Maggiore($N1, $N2)
{
  if($N1 > $N2)
    return $N1;
  else
    return $N2;
}
$M = Maggiore($_POST['Num1'], $_POST['Num2']);
print("il maggiore tra i due numeri e' : $M ");
?>
```



La via alternativa più immediata per riferirsi ai parametri passati da HTML sarebbe quella di non dover specificare a ogni riferimento gli array `$_GET` o `$_POST`. Nell'esempio precedente avremmo potuto scrivere solo `$Num1` per riferirci al primo parametro e non `$_POST['Num1']`.

Possiamo, cioè, fare riferimento al nome dato ai parametri utilizzando la **proprietà NAME** nel file HTML. Per fare questo occorre modificare in **on** l'impostazione **register_globals** nel file di configurazione `php.ini`.

L'array superglobale `$_REQUEST`

Utilizzando l'array associativo superglobale `$_REQUEST` è possibile ottenere informazioni sul contenuto degli array `$_GET` e `$_POST`.

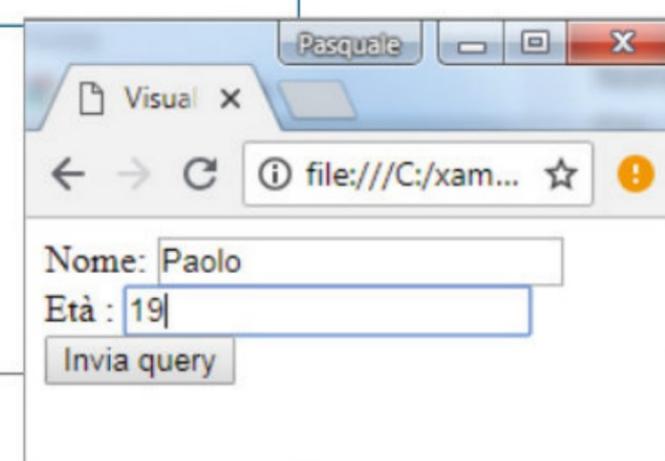
Specificando la chiave alfanumerica nell'array `$_REQUEST` si ottengono i valori dei parametri passati dal form sia attraverso il metodo GET, sia attraverso il metodo POST.
La sintassi da utilizzare è:

```
$_REQUEST["NomeParametro"]
```

dove *NomeParametro* è il nome dell'elemento INPUT di HTML del quale si vuole recuperare il valore. Vediamo subito un esempio in cui si visualizzano i valori di due elementi INPUT di HTML passati con il metodo POST.

File Visualizza.html

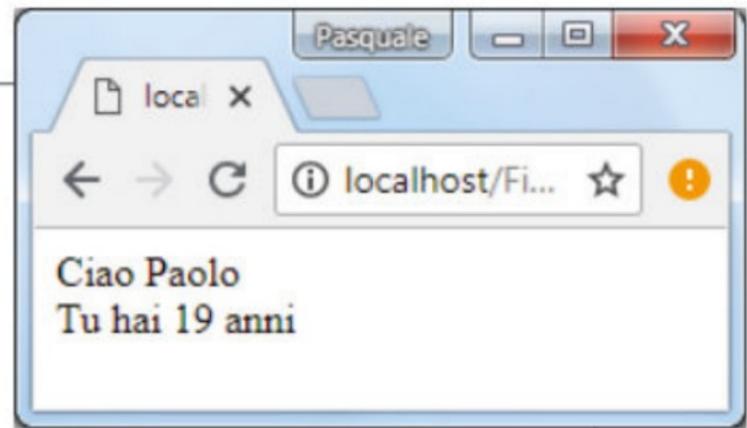
```
<html>
<body>
```



```
<form action="http://localhost/Visualizza.php" method="POST">
    Nome: <input type="text" name="Nome" />
    Età : <input type="text" name="Età" />
    <input type="submit" value="Invia query">
</form>
</body>
</html>
```

File Visualizza.php

```
<?php
if($_REQUEST["Nome"] && $_REQUEST["Età"] )
{
    echo "Ciao ".$_REQUEST['Nome']. "<br/>";
    echo "Tu hai ".$_REQUEST['Eta']. " anni";
    exit();
}
?>
```



La variabile predefinita **`$_PHP_SELF`**

Consideriamo l'esempio precedente e riscriviamolo in un unico file nel seguente modo:

File Visualizza2.php

```
<?php
    if($_REQUEST["Nome"] && $_REQUEST["Eta"] )
    {
        echo "Ciao ".$_REQUEST['Nome']. "<br/>";
        echo "Tu hai ". $_REQUEST['Eta'] . " anni";
        exit();
    }
?>
<html>
    <body>
        <form action=<?php $_PHP_SELF ?>" method="POST">
            Nome: <input type="text" name="Nome" />
            Eta : <input type="text" name="Eta" />
            <input type="submit" value="Invia query"/>
        </form>
    </body>
</html>
```

La variabile predefinita **`$_PHP_SELF`** contiene il nome dello script stesso in cui è stata chiamata. Ciò consente, non appena si fa clic sul pulsante *submit*, di richiamare lo stesso file PHP.

16 Interazione con ulteriori elementi del form

HTML si presenta in sostanza come un'estensione GUI per un programma lato server. Esaminiamo ora gli altri elementi che contribuiscono ad arricchire tale interfaccia grafica.

L'elemento SELECT

L'elemento **SELECT** di HTML permette di scegliere tra una lista di opzioni.

16 Interazione con ulteriori elementi del form

HTML si presenta in sostanza come un'estensione GUI per un programma lato server. Esaminiamo ora gli altri elementi che contribuiscono ad arricchire tale interfaccia grafica.

L'elemento SELECT

L'elemento **SELECT** di HTML permette di scegliere tra una lista di opzioni.

L'elemento **RADIO**

L'elemento **RADIO** ci permette di effettuare scelte mutuamente esclusive. Vediamo un esempio in cui occorre scegliere solo una opzione tra le 9 presenti, come mostra la figura riportata al termine della pagina che segue.

Il file HTML mostrato qui sotto rappresenta la pagina HTML statica relativa a quanto mostrato nella figura.

Pagina HTML statica (radio.html)

```
<form method="POST" action="Verifica.php">
    <h2> Indovina i numeri vincenti </h2>
    Scegli il primo numero (da 1 a 9) <br/>
    <input type="radio" name= "scelta1" value=1 />1
    <input type="radio" name= "scelta1" value=2 />2
    <input type="radio" name= "scelta1" value=3 />3
    <input type="radio" name= "scelta1" value=4 />4
    <input type="radio" name= "scelta1" value=5 />5
    <input type="radio" name= "scelta1" value=6 />6
    <input type="radio" name= "scelta1" value=7 />7
    <input type="radio" name= "scelta1" value=8 />8
    <input type="radio" name= "scelta1" value=9 />9
    <br/>Scegli il secondo numero (da 1 a 9) <br/>
    <input type="radio" name= "scelta2" value=1 />1
    <input type="radio" name= "scelta2" value=2 />2
    <input type="radio" name= "scelta2" value=3 />3
    <input type="radio" name= "scelta2" value=4 />4
    <input type="radio" name= "scelta2" value=5 />5
    <input type="radio" name= "scelta2" value=6 />6
    <input type="radio" name= "scelta2" value=7 />7
    <input type="radio" name= "scelta2" value=8 />8
    <input type="radio" name= "scelta2" value=9 />9
    <br/> <input type="submit" value="invia">
    <input type="reset " value="resetta">
</form >
```

Pagina HTML dinamica (radio.php)

```
<html>
<form method="POST" action="Verifica.php">
    <h2> Indovina la mia combinazione </h2>
    Scegli un numero da 1 a 9:
    <?php
        for($i=1;$i<10;$i++)
            echo "<input type=\"radio\""
                  name= \"scelta1\" value=$i />$i ";
    ?>
    <br/> Scegli un secondo numero:
    <?php
        for($i=1;$i<10;$i++)
            echo "<input type=\"radio\""
                  name= \"scelta2\" value=$i />$i ";
    ?>
    <br/>
    <input type="submit" value="invia" />
    <input type="reset" value="resetta" />
    </form>
</html>
```

Il file PHP richiamato da ambedue le pagine HTML è il seguente.

File Verifica.php

```
<?php  
    $com1=5; $scelta1 = $_POST['scelta1'];  
    $com2=9; $scelta2 = $_POST['scelta2'];  
    if (($scelta1 == $com1) && ($scelta2 == $com2))  
        print("Hai indovinato i due numeri $com1 $com2!");  
    elseif (($scelta1 == $com1) || ($scelta2 == $com2))  
        print("Hai indovinato solo un numero");  
    else  
        print("Hai sbagliato tutto!");  
?>
```

Un possibile output è mostrato nella figura.

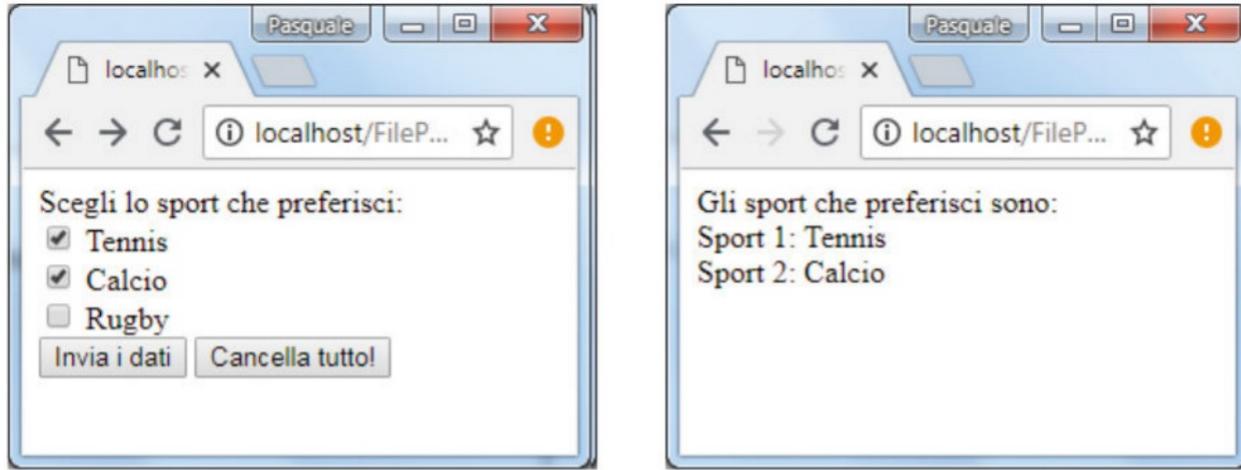


Parametri a più valori: l'elemento CHECKBOX

Le caselle di controllo, ovvero gli elementi CHECKBOX in HTML, permettono di scegliere più valori. Per essere sicuri che PHP riconosca i valori multipli che il browser passa a uno script, è necessario che il valore dell'attributo NAME termini con le parentesi quadre []. Il valore presente in `$_POST` o `$_GET` sarà un array invece di una semplice stringa.

Vediamo un esempio: consideriamo il seguente file HTML tramite il quale un utente deve comunicare quali sono i suoi sport preferiti.

```
<html>
    <form method="POST" action="http://localhost/FilePersonalI/Sport.php">
        Scegli lo sport che preferisci: <br/>
        <input type="CHECKBOX" name="sport[]" value="Tennis" />
            Tennis <br/>
        <input type="CHECKBOX" name="sport[]" value="Calcio" />
            Calcio<br/>
        <input type="CHECKBOX" name="sport[]" value="Rugby" />
            Rugby <br/>
        <input type="submit" value="Invia i dati">
        <input type="reset" value="Cancella tutto!">
    </form>
</html>
```



Il file PHP richiamato è il seguente.

File Sport.php

```
<?php
    $sp = $_POST['sport'];
    echo "Gli sport che preferisci sono: <br/>";
    $cnt = count($sp);
    for($i=0; $i < $cnt; $i++)
    {
        echo "Sport ";
        echo $i+1;
        echo ": $sp[$i] <br/>";
    }
?>
```

OSSERVA COME SI FA

- Passare parametri tra file PHP (link HREF)

Creiamo, in PHP, una pagina HTML che mostri un elenco di alcune regioni italiane con il rispettivo capoluogo. Facendo clic su un apposito link, posto accanto a ogni regione, deve essere possibile visualizzare tutte le altre province di quella regione. La situazione è mostrata nelle figure seguenti. La figura di destra mostra il risultato di ciò che accade facendo clic sull'apposito link sulla riga relativa alla regione Puglia.



Supponiamo di aver memorizzato in un array multidimensionale le regioni e, per ogni regione, le province. Costruiamo l'elenco all'interno di una tabella HTML e creiamo i link utilizzando gli elementi **href** di HTML. Di seguito è mostrato il codice relativo al file *Regioni.php*. Gli elementi *href* presenti in questo file richiamano il file *Province.php* passandogli come parametro il nome della regione di cui si vogliono visualizzare le province. Il file *Province.php* richiamato all'interno dell'elemento *href* mostra le province relative alla regione scelta. Il parametro di nome *Reg* conterrà il nome della regione.

File Regioni.php

```
<?php
$Regioni = array(array('Lombardia', 'Milano', 'altre province'),
    array('Puglia', 'Bari', 'altre province'),
    array('Trentino', 'Trento', 'altre province'),
    array('Veneto', 'Venezia', 'altre province')
);
print("<h2>Regioni italiane</h2>");
print("<table style=\"border-style:double;
    border-width:4pt; border-color:black\">");
print("<tr style=\"font-weight:rockwell\">");
print("<td>Nome </td>");
print("<td>Capoluogo</td>");
print("<td></td></tr>");
for($i = 0; $i < count($Regioni); $i++ )
{
    print"<tr>";
    for($j = 0; $j < count($Regioni[$i]); $j++ )
        if($j!=2)
            print"<td>" . $Regioni[$i][$j]. " </td>";
        else
            print"<td><a href=\"Province.php?Reg=". 
                $Regioni[$i][0]. "\">". $Regioni[$i][$j]. "</a></td></tr>\"";
}
print "</table>";
?>
```

File Province.php

```
<?php
$Province = array( array( 'Lombardia', 'Milano', 'Bergamo', 'Brescia'....),
array('Puglia', 'Bari', 'Brindisi', 'Foggia', 'Lecce', 'Taranto'.... ),
array('Trentino', 'Trento', ...),
array('Veneto', 'Venezia', 'Vicenza', ... )
);
print("<h3>Province della regione ". $_REQUEST['Reg']. "</h3>");
print("<table style=\"border-style:double; border-width:
        4pt; border-color:black\">");
print("<tr style=\"font-weight:rockwell\>\"");
print("<td> Nome </td>");
for($i = 0; $i < count($Province); $i++)
{
    for($j = 1; $j < count($Province[$i]); $j++)
        if($Province[$i][0]== $_REQUEST['Reg'])
            print"<tr><td>" . $Province[$i][$j]. " </td></tr>";
}
print"</table>";
print("<a href=\"Regioni.php\> Torna alle regioni </a>");

?>
```

Il file *Province.php* può essere riscritto **inframmezzando** codice HTML con codice PHP, come è mostrato di seguito. Ogni volta che compare codice PHP, si aprono e si chiudono i relativi tag. Per inserire il valore di una variabile PHP all'interno di codice HTML, abbiamo utilizzato la sintassi:

```
<?=NomeVariabile ?>
```

Da notare l'uso del carattere =.

File Province.php

```
<?php
$Province = array( array( 'Lombardia', 'Milano', 'Bergamo', 'Brescia', ...),
                   array( 'Puglia', 'Bari', 'Brindisi', 'Foggia', 'Lecce', 'Taranto'),
                   array( 'Trentino', 'Trento', ...),
                   array( 'Veneto', 'Venezia', 'Vicenza', ... )
);
?>
<h3>Province della regione <?=$_REQUEST['Reg']?>
</h3>
<table style="border-style:double; border-width:4pt; border-color:black">
    <tr style="font-weight:rockwell">
        <td> Nome</td>
        <?php
            for($i = 0; $i < count($Province); $i++)
            {
                for($j = 1; $j < count($Province[$i]); $j++)
                    if($Province[$i][0]== $_REQUEST['Reg'])
                    {
                        ?>
                        <tr><td><?=$Province[$i][$j]?> </td></tr>
                        <?php
                    }
            }
        ?>
    </table>
    <a href="Regioni.php"> Torna alle regioni </a>
```

17 Non solo HTML

Riconsideriamo l'ultimo esempio. Il risultato dell'elaborazione è stato visualizzato in modo molto semplice, badando solo al contenuto della comunicazione e senza aggiungere dettagli riguardo alla formattazione o all'aspetto grafico. Ora vogliamo riscrivere il file PHP in modo da tenere presenti anche questi aspetti.

Mettere insieme PHP, HTML, CSS e script lato client

Quando si prepara la pagina HTML risultato di un'elaborazione PHP, è possibile inserire all'interno di quest'ultima anche degli script lato client. Ciò che importa, infatti, è ottenere una pagina che sia leggibile da un browser, così come lo è una pagina ricca di script lato client. In questo modo è possibile **inserire un codice di qualsiasi linguaggio di scripting lato client** (ad esempio JavaScript) in modo dinamico, cioè a tempo di esecuzione dello script PHP. Riprendiamo l'esempio precedente del maggiore tra due numeri e modifichiamolo aggiungendo alla pagina HTML risultato un pulsante che, quando viene premuto, visualizza il quadrato del numero maggiore calcolato da PHP.

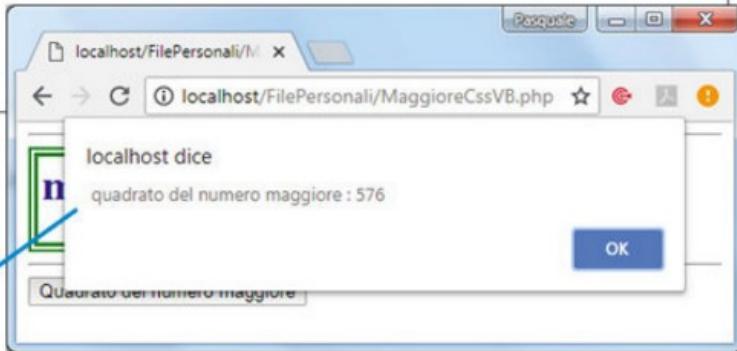
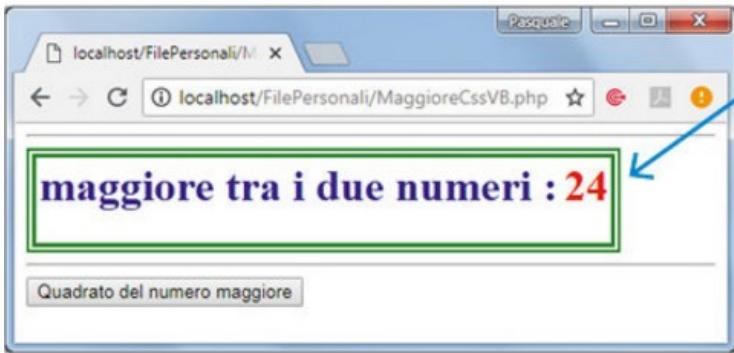
Creazione dinamica di una pagina HTML contenente elementi CSS e script lato client

Elementi CSS per la presentazione grafica del risultato

Script lato client (funzione JavaScript)

```
<?php
    function Maggiore($N1, $N2)
    {
        if ($N1 > $N2)
            return $N1;
        else
            return $N2;
    }
    $M = Maggiore($_POST['Num1'], $_POST['Num2']);           // (1)
    print("<html>");
    print("<head>");
    print("<style type=\\"TEXT/CSS\\"> ");
    print("table.Risultato {border-style:double; border-width:5pt;
        border-color:green}");
    print("h1.primo {color:blue}");
    print("h1.secondo {color:red}");
    print("</style>");
    print("<script Language=\\"Javascript\\">");
    print("function Somma( )");
    print("{");
    print("alert(\\"quadrato del numero maggiore: ". $M*$M. \")");
    print("}");
    print("</script>");
    print("</head>");
    print("<body>");
    print("<hr>");
    print("<table class=\\"Risultato\\">");
    print("<tr>");
    print("<td>");
    print("<h1 class=\\"primo\\"> maggiore tra i due numeri: </h1>");
    print("</td>");
    print("<td>");
    print("<h1 class=\\"secondo\\"> $M </h1>");
    print("</td>");
    print("</tr>");
```

```
print("</table>");  
print("<hr/>");  
print("<form>");  
print("<input type=\"button\" value=\"Quadrato del numero maggiore\"  
      onClick=\"Somma()\">");  
  
print("</form>");  
print("</body>");  
print("</html>");  
?  
?>
```



Rifletti

Osserva la sequenza di istruzioni `print` presenti nel codice precedente. Lo stesso risultato si sarebbe potuto ottenere creando, per concatenazione, un'unica stringa contenente tutto il codice HTML e poi visualizzandola con una sola istruzione `print`.

OSSERVA COME SI FA

1. Visualizzare una lista di articoli e relative fotografie

Creiamo una programma PHP che lavori su un array multidimensionale in cui è memorizzata una lista di articoli. Per ogni articolo si conoscono il codice, le caratteristiche principali e il nome del file (completo di percorso) contenente una sua fotografia. Il programma deve visualizzare la lista degli articoli, comprese le fotografie, in una tabella HTML.

Analisi del problema

Con una serie di *print* impostiamo l'intestazione della tabella HTML. Utilizziamo, poi, due cicli *for* annidati per scorrere le righe, e all'interno di ogni riga le celle che la compongono. Quando la variabile che scorre le celle della riga (*\$j*) varrà 2, creeremo dinamicamente un elemento IMG di HTML che avrà come proprietà SRC il contenuto dell'array PHP in terza posizione, cioè il nome del file contenente l'immagine. Il risultato finale è mostrato nella figura al termine del codice.

```
<html>
<head>
<style>
table {
    border: 3px solid blue;
    border-collapse: separate;
    border-spacing: 5px;
}
td {
    border: 1px solid #999;
    padding: 0.5rem;
}
th {
    background: lightblue;
    border-color: white;
}
</style>
</head>
<body>
<?php
$Articoli = array(
    // (1)
```

```

array('Art01', 'tablet ', 'art1.jpg'),
array('Art02', 'web cam', 'art2.jpg'),
array('Art03', 'Mini proiettore', 'art3.jpg'),
array('Art04', 'Netbook', 'art4.jpg'));
print("<h2>Articoli per la vendita</h2>");
print("<table >"); // (2)
print("<tr >"); // (3)
print("<th>Codice</th>"); // (4)
print("<th>Caratteristiche</th>"); // (5)
print("<th>Immagine</th></tr>"); // (6)
for($i = 0; $i < count($Articoli); $i++) // (7)
{
    print "<tr>";
    for($j = 0; $j < count($Articoli[$i]); $j++) // (8)
        if($j != 2)
            print "<td> " . $Articoli[$i][$j]. " </td>";
        else
            print("<td></td>"); // (9)
            print "</tr>";
}
?>
</table>
</body>
</html>

```

Elemento IMG creato
dinamicamente. La
proprietà SRC prende il
valore dell'array PHP



2. Visualizzare una lista di articoli e relative fotografie solo su richiesta

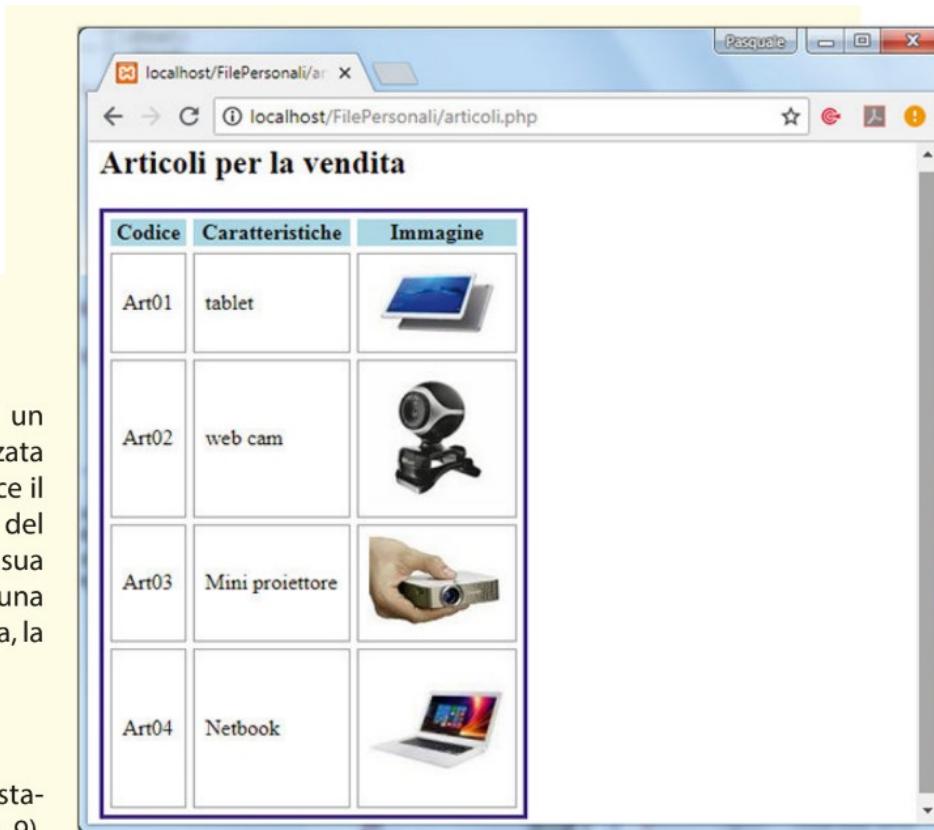
Creiamo un programma PHP che lavori su un array multidimensionale in cui è memorizzata una lista di articoli. Per ogni articolo si conosce il codice, le caratteristiche principali e il nome del file (completo di percorso) contenente una sua fotografia. Il programma deve visualizzare in una tabella HTML la lista degli articoli e, su richiesta, la foto di ogni articolo.

Analisi del problema

Con una serie di *print* impostiamo l'intestazione della tabella HTML (istruzioni da 7 a 9).

Utilizziamo, poi, due *for* annidati per scorrere le righe, e all'interno di ogni riga, le celle che la compongono (istruzioni 10 e 11). Quando la variabile che scorre le celle della riga (*\$j*) varrà 2, creeremo dinamicamente un elemento INPUT di tipo *button* (12).

A tale pulsante associeremo l'evento *onClick()* richiamando la funzione Javascript *Cambia()* (1) e passando come parametro il contenuto dell'array PHP in terza posizione, cioè il nome del file contenente l'immagine (3). La funzione Javascript, precedentemente definita, modificherà la proprietà SRC dell'unica immagine presente nel documento HTML attraverso l'opportuna istruzione JavaScript (2).



The screenshot shows a web browser window titled "localhost/FilePersonali/articoli.php". The page has a header "Articoli per la vendita". Below it is a table with three columns: "Codice", "Caratteristiche", and "Immagine". The table contains four rows of data:

Codice	Caratteristiche	Immagine
Art01	tablet	
Art02	web cam	
Art03	Mini proiettore	
Art04	Netbook	

```

<html>
<head>
<style>
table {
    border: 3px solid blue;
    border-collapse: separate;
    border-spacing: 5px;
}
td {
    border: 1px solid #999;
    padding: 0.5rem;
}
th {
    background: lightblue;
    border-color: white;
}
</style>
<script Language="Javascript">
function Cambia(Str)
{
    document.F1.Immagine.src=Str;
}
</script>
</head>
<body>
<body>
<form name="F1">
<?php
$Articoli = array(
    array('Art01', 'tablet', 'art1.jpg'),
    array('Art02', 'web cam', 'art2.jpg'),
    array('Art03', 'Mini proiettore', 'art3.jpg'),
    array('Art04', 'Netbook', 'art4.jpg'));
print("<h2>Articoli per la vendita</h2>");
print("<table >"); // (1)
// (2)
print("<tr >"); // (3)
print("<th>Codice</th>"); // (4)
print("<th>Caratteristiche</th>"); // (5)
print("<th>Immagine</th></tr>"); // (6)
for($i=0; $i < count($Articoli); $i++) // (7)
{
    print ("<tr>"); // (8)
    for ( $j=0; $j < count($Articoli[$i]); $j++) // (9)
        if($j==2) // (10)
            print("<td> <input type = \"button\" value=\"Visualizza \" onClick=\"Cambia(\"".$Articoli[$i][$j]."\")\"></td>"); // (11)
        else // (12)
            print("<td> " . $Articoli[$i][$j] . "</td>"); // (13)
    print("</tr>"); // (14)
}
?>
</table>
Immagine articolo scelto

</form>
</body>
</html>

```

Articoli per la vendita

Codice	Caratteristiche	Immagine
Art01	tablet	Visualizza
Art02	web cam	Visualizza
Art03	Mini proiettore	Visualizza
Art04	Netbook	Visualizza



Immagine articolo scelto

Occorre prestare molta attenzione al modo in cui è stata costruita l'istruzione (12):

```

print("<td> <input type = \"button\" value=\"Visualizza \" onClick=\"Cambia(\"".$Articoli[$i][$j]."\")\"></td>");
```

IL LINGUAGGIO PHP

 MAPPA
MODIFICABILE

