

Benvenuti in PHP

PHP: Hypertext Preprocessor (**PHP**) è un linguaggio di scripting gratuito, molto popolare e open source. Gli script PHP vengono eseguiti sul **server** **Prima** .

Solo una breve lista di ciò che PHP è in grado di fare:

- Generare contenuti dinamici per le pagine
- Creare, aprire, leggere, scrivere, cancellare e chiudere file sul server
- Raccogliere i dati dei moduli
- Aggiungere, cancellare e modificare le informazioni memorizzate nel vostro database
- controllare l'accesso degli utenti
- crittografare i dati
- e molto altro ancora!

iniziare questo tutorial, dovresti avere una comprensione di base dell'**HTML**.

PHP ha abbastanza potenza per lavorare nel cuore di **WordPress**, il sistema di blogging più trafficato del web. Ha anche il grado di profondità necessario per far funzionare **Facebook**, il più grande social network del web!

Perché PHP

PHP **funziona** su numerose e diverse piattaforme, tra cui Windows, Linux, Unix, Mac OS X, e così via.

PHP è **compatibile** con quasi tutti i server moderni, come Apache, IIS e altri.

PHP **supporta** una vasta gamma di database.

PHP è **gratuito**!

PHP è facile da imparare e funziona in modo efficiente sul lato server.

Sintassi PHP

Uno script PHP inizia con `<?php` e finisce con `?>` :
`<?php` // il codice PHP va qui `?>`

Ecco un esempio di un semplice file PHP. Lo script PHP usa una funzione integrata chiamata "echo" **per** inviare il testo "Hello World!" a una pagina web.

```
<html>
  <head>
    <title>My First PHP Page</title>
  </head>
<body>
  <php? echo "Hello World!"; ?>
</body>
</html>
```

Le dichiarazioni PHP terminano con il **punto e virgola (;)**.

In alternativa, possiamo includere PHP nel **tag**

HTML **<script>**.

```
<html>
<head>
<title>My First PHP Page</title>
</head>
  <body>
    <script language="php"> echo "Hello World!"; </script>
  </body>
</html>
```

Tuttavia, l'ultima versione di PHP rimuove il supporto per i tag `<script language="php">`. Come tale, raccomandiamo di usare esclusivamente `<?php ?>`.

Puoi anche usare i tag sintetici PHP, `<? ?>`, purché siano supportati dal server.

```
<?
  echo "Hello World!";
?>
```

Tuttavia, `<?php ?>` , **come** standard ufficiale, è il modo raccomandato per definire gli script PHP.

Echo

PHP ha una funzione integrata "echo", che è usata per emettere del testo.

In realtà, non è una funzione; è un costrutto del linguaggio. Come tale, non richiede parentesi.

Emettiamo un testo.

```
<?php echo "This is a text." ?>
```

Il testo dovrebbe essere tra **virgolette** singole o doppie.

Dichiarazioni PHP

Ogni dichiarazione PHP deve finire con un **punto e virgola**

```
<?php  
echo "A";  
echo "B";  
echo "C";  
?>
```

Dimenticarsi di aggiungere un punto e virgola alla fine di una dichiarazione comporta un **errore**.

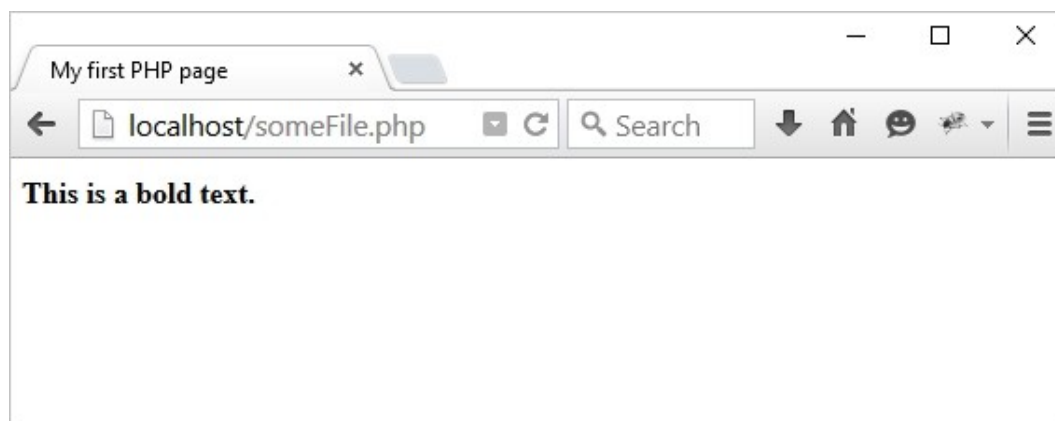
Echo

Il markup HTML può essere aggiunto al testo nell'istruzione **echo**.

```
<?php  
echo "<strong>This is a bold text.</strong>";  
?>
```

PHP

Risultato:



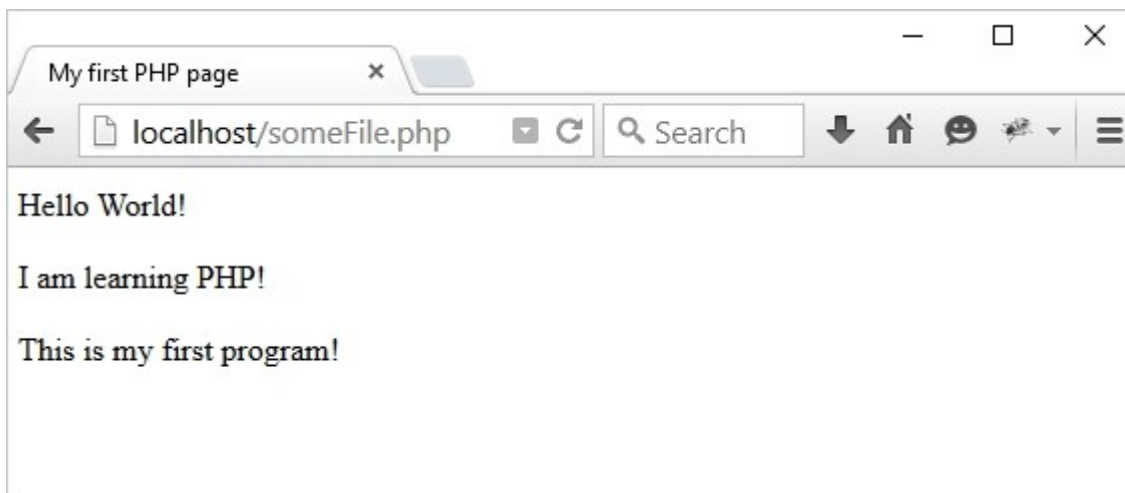
Commenti

Nel codice PHP, un **commento** è una linea che non viene eseguita come parte del programma. Potete usare i commenti per comunicare agli altri in modo che capiscano cosa state facendo, o per ricordare a voi stessi cosa avete fatto.

Un commento **a linea singola** inizia con `//`:

```
<?php
    echo "<p>Hello World!</p>";
    // This is a single-line comment
    echo "<p>I am learning PHP!</p>";
    echo "<p>This is my first program!</p>";
?>
```

Risultato:



Commenti a più righe

I commenti multilinea si usano per comporre commenti che occupano più di una singola linea.

Un commento multilinea inizia con `/*` e finisce con `*/`.

```
<?php
    echo "<p>Hello World!</p>";
    /*
    Questo è un
    commento che va su
    più righe!
    */
    echo "<p>I am learning PHP!</p>";
    echo "<p>This is my first program!</p>";
?>
```

Aggiungere commenti mentre si scrive il codice è una buona pratica. Aiuta gli altri a capire il vostro pensiero e vi rende più facile ricordare i vostri processi di pensiero quando fate riferimento al vostro codice in seguito.

Variabili

Variabili sono usate come "contenitori" in cui immagazziniamo informazioni.

Una variabile PHP inizia con un segno di dollaro (\$), che è seguito dal nome della variabile.

```
$variabile_nome = valore;
```

PHP

Regole per le variabili PHP:

- Un nome di variabile deve iniziare con una lettera o un trattino basso
- Un nome di variabile non può iniziare con un numero
- Un nome di variabile può contenere solo caratteri alfanumerici e underscore (A-z, 0-9, e _)
- I nomi delle variabili sono sensibili alle maiuscole e alle minuscole (\$name e \$NAME sarebbero due variabili diverse):

```
<?php
```

```
    $name = 'John';
```

```
    $age = 25;
```

```
    echo $name;
```

```
// Outputs 'John'
```

```
?>
```

Nell'esempio qui sopra, notate che non abbiamo dovuto dire a PHP che tipo di dati è la variabile.

PHP converte automaticamente la variabile nel tipo di dati corretto, a seconda del suo valore.

A differenza di altri linguaggi di programmazione, PHP non ha un comando per dichiarare una variabile. Viene creata nel momento in cui si assegna un valore ad essa per la prima volta.

Costanti

Costanti sono simili alle variabili eccetto che non possono essere cambiate o indefinite dopo che sono state definite.

Iniziate il nome della vostra costante con una lettera o un trattino basso.

Per creare una costante, usate la funzione **define()**:

```
define(nome, valore, case-insensitive)
```

Parametri:

nome: Specifica il nome della costante;

valore: Specifica il valore della costante;

case-insensitive: Specifica se il nome della costante deve essere case-insensitive. Il valore predefinito è **false**;

l'esempio seguente crea una costante con un nome **case-sensitive**:

```
<?php
```

```
    define("MSG", "Hi Students!");
```

```
    echo MSG;
```

```
    // Outputs "Hi Students!" ?>
```

L'esempio seguente crea una costante con un nome **insensibile alle maiuscole e alle minuscole**:

```
<?php
```

```
define("MSG", "Hi Students!", true);?>
```

Non è necessario il segno del dollaro (\$) prima del nome della costante.

Tipi di dati

Le variabili possono memorizzare una varietà di tipi di dati.

Tipi di dati supportati da PHP: **String**, **Integer**, **Float**, **Boolean**, **Array**, **Object**, NULL, Resource.

Stringa PHP

Una stringa è una sequenza di caratteri, come "Ciao mondo!".

Una stringa può essere qualsiasi testo all'interno di un insieme di **apici è una sequenza di caratteri, come "Ciao mondo!"**.

singoli o doppi.

```
<?php $stringa1 = "Ciao mondo!"; //doppi apici $stringa2?>
```

Puoi unire due stringhe insieme usando l'operatore di concatenazione punto (.)

Per esempio: **echo \$s1 . \$s2**

Intero PHP

Un **numero intero** è un numero intero (senza decimali) che deve soddisfare i seguenti criteri:

- Non può contenere virgole o spazi vuoti
- Non deve avere un punto decimale
- Può essere sia positivo che negativo

```
<?php
    $int1 = 42;
    // numero positivo $int2
?>
```

Le variabili possono memorizzare una varietà di tipi di dati.

PHP Float

Un **float**, o numero a virgola mobile, è un numero che include un punto decimale.

```
<?php
    $x = 42.168;
?>
```

PHP booleano

Un booleano rappresenta due possibili stati: VERO o FALSO.

```
<?php
$x = true; $y = false;
?>
PHP
```

I booleani sono spesso usati nei test condizionali, che saranno trattati più avanti nel corso.

La maggior parte dei tipi di dati può essere usata in combinazione tra loro. In questo esempio, **stringa** e **intero** sono messi insieme per determinare la somma di due numeri.

```
<?php
$str = "10";
$int = 20;
$sum = $str + $int;
echo ($sum);
// ris 30
?>
```

PHP converte automaticamente ogni variabile nel tipo di dati corretto, secondo il suo valore. Questo è il motivo per cui la variabile **\$str** è trattata come un numero nell'aggiunta.

Variabile di Variabili

Con PHP, potete usare una variabile per specificare il nome di un'altra variabile.

Perciò, una **variabile di variabile** tratta il valore di un'altra variabile come il suo nome. **Per**

esempio:

```
<?php
$a = 'hello';
$hello = "Hi!";
echo $$a ;
```

Scrivo hello!

\$\$a è una variabile che usa il valore di un'altra variabile, **\$a**, come nome. Il valore di **\$a** è uguale a "hello". La variabile risultante è **\$hello**, che contiene il valore "Hi!".

Operatori

Gli **operatori** eseguono operazioni su variabili e valori.

$$\begin{array}{ccccccc} 1 & + & 2 & = & 3 \\ \text{operand} & \text{operator} & \text{operand} & \text{operator} & \text{operand} \end{array}$$

Operatori aritmetici

Gli operatori aritmetici lavorano con valori numerici per eseguire operazioni aritmetiche comuni.

Operator	Name	Example
+	Addition	\$x + \$y
-	Subtraction	\$x - \$y
*	Multiplication	\$x * \$y
/	Division	\$x / \$y
%	Modulus	\$x % \$y

Esempio:

```
<?php
```

```
$num1 = 8;
```

```
$num2 = 6;
```

```
//Addition
```

```
echo $num1 + $num2; //14echo
```

```
//Subtraction
```

```
$num1 - $num2; //2echo
```

```
//Multiplication
```

```
$num1 * $num2; //48echo
```

```
//Division
```

```
$num1 / $num2; //1.33333333333
```

```
?>
```

Modulo

L'operatore **modulo**, rappresentato dal segno %, restituisce il resto della divisione del primo operando per il secondo operando:

```
<?php
$x = 14;
$y = 3;
echo $x % $y; // 2
?>
PHP
```

Se usate numeri in virgola mobile con l'operatore di modulo, essi saranno convertiti in **interi** prima dell'operazione.

Incremento e decremento

Gli operatori di **incremento** sono usati per incrementare il valore di una variabile.
e **decremento** sono usati per decrementare il valore di una variabile.

`$x++;` // equivalent to `$x = $x+1;`

`$x--;` // equivalente a `$x = $x-1;`

PHP

Gli operatori di incremento e decremento precedono o seguono una variabile.

`$x++;` // post-increment

`$x--;` // post-decrement

`++$x;` // pre-increment

`--$x;` // pre-decremento

PHP

La differenza è che il post-incremento restituisce il valore originale **prima di** cambiare la variabile, mentre il pre-incremento cambia prima la variabile e poi restituisce il valore. **Esempio:**

`$a = 2; $b = $a++; // $a=3, $b=2`

`$a = 2; $b = ++$a; // $a=3, $b=3`

Gli operatori di incremento sono usati per incrementare il valore di una variabile.

Operatori di assegnazione

Gli operatori di **assegnazione** sono usati per scrivere valori nelle variabili.

```
$num1 = 5;
```

```
$num2 = $num1;
```

PHP

\$num1 e **\$num2** **Le** now contain the value of 5.

assegnazioni possono anche essere usate insieme agli operatori aritmetici.

Assignment	Same as...	Description
$x+=y$	$x = x + y$	Addition
$x-=y$	$x = x - y$	Subtraction
$x*=y$	$x = x * y$	Multiplication
$x/=y$	$x = x / y$	Division
$x\%=y$	$x = x \% y$	Modulus

Esempio:

```
<?php
```

```
$x = 50;
```

```
$x += 100;
```

```
echo $x;
```

```
// Uscite: 150
```

```
?>
```

Operatori di confronto

Gli operatori confrontano due valori (numeri o stringhe).

Gli operatori di confronto sono usati all'interno di dichiarazioni condizionali e valutano **TRUE** o **FALSE**. Gli operatori confrontano due valori (numeri o stringhe).

Operator	Name	Example	Result
<code>==</code>	Equal	<code>\$x == \$y</code>	Returns true if \$x is equal to \$y
<code>===</code>	Identical	<code>\$x === \$y</code>	Returns true if \$x is equal to \$y , and they are of the same type
<code>!=</code>	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
<code><></code>	Not equal	<code>\$x <> \$y</code>	Returns true if \$x is not equal to \$y
<code>!==</code>	Not identical	<code>\$x !== \$y</code>	Returns true if \$x is not equal to \$y , or they are not of the same type

Fate attenzione ad usare `==` e `===` ; il primo non controlla il tipo di dati.

Operatori di confronto aggiuntivi:

Operator	Name	Example	Result
<code>></code>	Greater than	<code>\$x > \$y</code>	Returns true if \$x is greater than \$y
<code><</code>	Less than	<code>\$x < \$y</code>	Returns true if \$x is less than \$y
<code>>=</code>	Greater than or equal to	<code>\$x >= \$y</code>	Returns true if \$x is greater than or equal to \$y
<code><=</code>	Less than or equal to	<code>\$x <= \$y</code>	Returns true if \$x is less than or equal to \$y

Gli operatori di confronto di PHP sono utilizzati per confrontare due valori (numero o stringa).

Operatori Logici

Gli **operatori logici** sono utilizzati per verificare che una o più le condizioni siano vere.

Operator	Name	Example	Result
and	And	\$x and \$y	True if both \$x and \$y are true
or	Or	\$x or \$y	True if either \$x or \$y is true
xor	Xor	\$x xor \$y	True if either \$x or \$y is true, but not both
&&	And	\$x && \$y	True if both \$x and \$y are true
	Or	\$x \$y	True if either \$x or \$y is true
!	Not	!\$x	True if \$x is not true

- and (&&) verifica che entrambe le condizioni siano vere
- or (||) verifica che almeno una delle condizioni sia vera
- xor Vero solo se una delle condizioni è vera, ma non entrambe
- ! negazione, Vero se falso e Falso se vero

Array

Un array è una variabile speciale, che può contenere più di un valore alla volta.

Se avete una lista di elementi (una lista di nomi, per esempio), memorizzarli in singole variabili sarebbe così:

```
$name1 = "David";  
$name2 = "Amy";  
$name3 = "John";
```

Ma cosa succede se hai 100 nomi sulla tua lista? La soluzione: Crea un **array**!

Array numerici

Gli array numerici o indicizzati associano un indice numerico ai loro valori.

L'indice può essere assegnato automaticamente (l'indice inizia sempre da **0**), così:

```
$names = array("David", "Amy", "John");  
PHP
```

In alternativa, puoi assegnare il tuo indice manualmente.

```
$names[0] = "David";  
$names[1] = "Amy";  
$names[2] = "John";  
PHP
```

Abbiamo definito un array chiamato **\$names** that stores three values.

È possibile accedere agli elementi dell'array attraverso i loro indici.

```
echo $names[1]; // Stampa "Amy"
```

Ricordate che il primo elemento di una matrice ha l'indice **0**, non 1.

Potete avere interi, stringhe e altri tipi di dati insieme in un array.

Esempio:

```
<?php  
$myArray[0] = "John";  
$myArray[1] = "<strong>PHP</strong>";  
$myArray[2] = 21;  
  
echo "$myArray[0] ha $myArray[2] anni e conosce il $myArray[1]";  
  
// Outputs "John ha 21 anni e conosce il PHP"  
?>
```

Array associativi

Gli Array Associativi sono array che usano chiavi nominate che voi assegnate loro. Ci sono due modi per creare un array associativo:

```
$people = array("David"=>"27", "Amy"=>"21", "John"=>"42");  
// oppure  
$people['David'] = "27";  
$people['Amy'] = "21";  
$people['John'] = "42";
```

Nel primo esempio, notate l'uso dei segni => nell'assegnare i valori alle chiavi nominate. Usate le chiavi nominate per accedere ai membri dell'array.

```
$people = array("David"=>"27", "Amy"=>"21", "John"=>"42");  
echo $people['Amy'];
```

Array multidimensionali

Un **array multidimensionale** contiene uno o più array.

La dimensione di una matrice indica il numero di indici necessari per selezionare un elemento.

- Per una matrice **bidimensionale** array, avete bisogno di due indici per selezionare un elemento

- Per una matrice **tridimensionale**, avete bisogno di tre indici per selezionare un elemento

Gli array con più di tre livelli di profondità sono difficili da gestire.

Creiamo una matrice bidimensionale che contiene 3 matrici:

```
$people = array(  
    'online'=>array('David', 'Amy'),  
    'offline'=>array('John', 'Rob', 'Jack'),  
    'away'=>array('Arthur', 'Daniel')  
);
```

Ora l'array bidimensionale **\$people** contiene 3 array, e ha due indici: **row** e **column** . accedere agli elementi dell'array \$people, dobbiamo puntare ai due indici.

```
echo $people['online'][0]; //Outputs "David"
```

```
echo $people['away'][1]; //Output "Daniel"
```

Gli array nell'array multidimensionale possono essere sia numerici che associativi.

Istruzioni condizionali

Istruzioni condizionali eseguono azioni diverse per decisioni diverse.

L'istruzione **if else** è usata per eseguire un certo codice se una condizione è **vera**, e un altro codice se la condizione è **falsa**.

Sintassi:

```
if (condizione) {  
    da eseguire se la condizione è vera  
} else {  
    da eseguire se la condizione è falsa;  
}
```

Potete anche usare l'istruzione **if** senza l'istruzione **else**, se non avete bisogno di fare nulla, nel caso in cui la condizione sia **falsa**.

If Else

L'esempio seguente produrrà il numero più grande dei due.

```
<?php  
$x = 10;  
$y = 20;  
if ($x >= $y)  
    echo "x è maggiore o uguale a y";  
} else {  
    echo "y è maggiore di x";  
}  
?>
```


L'istruzione Elseif

Usate l'istruzione **if...elseif...else** per specificare una **nuova** condizione da testare, se la prima condizione è **falsa**.

Sintassi:

```
if (condizione) {  
    codice da eseguire se la condizione è vera  
} elseif (condizione) {  
    codice da eseguire se la condizione è vera  
} else {  
    codice da eseguire se la condizione è falsa  
}
```

Potete aggiungere tutte le dichiarazioni **elseif** che volete. Notate solo che l'istruzione **elseif** deve iniziare con un'istruzione **if**.

L'istruzione Elseif

Per esempio:

```
<?php  
$age = 21;  
if ($age < 13){  
    echo "Bambino";  
} elseif ($age > 13 && $age < 19) {  
    echo "Teenager";  
} else {  
    echo "Adult";  
}  
  
//Outputs "Adult"  
?>
```

Abbiamo usato l'**AND logico (&&)** per combinare le due condizioni e controllare se \$age è tra 13 e 19. Le parentesi graffe possono essere omesse se c'è solo una dichiarazione dopo l'**if-elseif-else**. **Per esempio:**

```
if($age <= 13)  
    echo "Child";  
else  
    echo "Adult";
```

Cicli

Quando si scrive codice, si può desiderare che lo stesso blocco di codice venga eseguito più e più volte. Invece di aggiungere diverse linee di codice quasi uguali in uno script, possiamo usare i **cicli** per eseguire un compito come questo.

Il ciclo **while** esegue un blocco di codice finché la condizione specificata è **vera**. **Sintassi:**

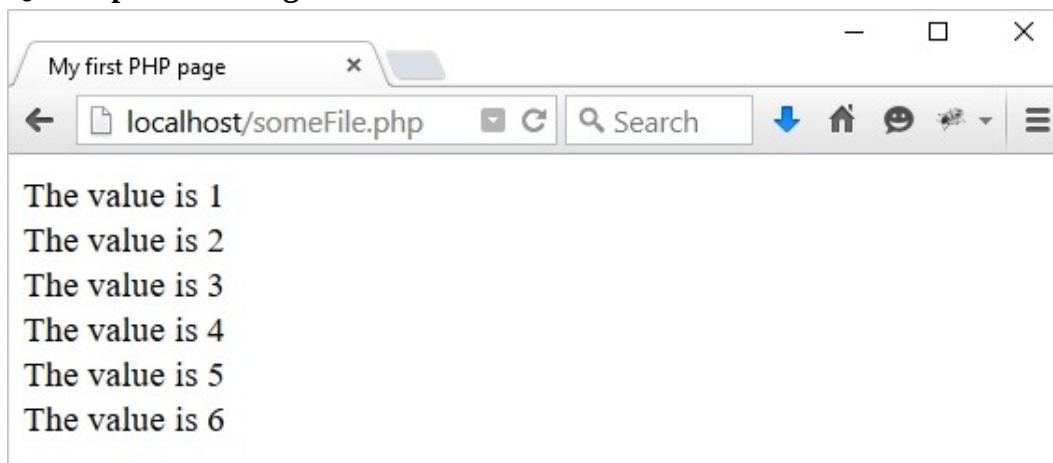
while (la condizione è **vera**) {}

Se la condizione non diventa mai **falsa**, l'istruzione continuerà ad essere eseguita indefinitamente.

L'esempio qui sotto imposta prima una variabile \$i a uno (\$i = 1). Poi, il ciclo **while** viene eseguito finché \$i è inferiore a sette (\$i < 7). \$i aumenterà di uno ad ogni esecuzione del ciclo (\$i++):

```
<?php
$i=1;
while($i<7){
    echo "The value is " . $i++;
}
?>
```

Questo produce il seguente risultato:



Il ciclo do...while

Il ciclo **do...while** esegue sempre il blocco di codice una volta, controlla la condizione e ripete il ciclo finché la condizione specificata è vera. **Sintassi:**

```
do {  
code to be executed;  
} while (la condizione è vera);
```

Indipendentemente dal fatto che la condizione sia **vera** o **falsa**, il codice sarà eseguito almeno **una volta**, il che potrebbe essere necessario in alcune situazioni.

L'esempio qui sotto scriverà un po' di output, e poi incrementerà la variabile \$i di uno. Poi la condizione viene controllata, e il ciclo continua a funzionare finché \$i è minore o uguale a 7.

```
$i = 5;  
do {  
echo "The number is " . $i . "<br/>";  
$i++;  
} while($i <= 7);
```

```
//Output  
//The number is 5  
//The number is 6  
//Il numero è 7
```

Notate che in un ciclo **do while**, la condizione viene testata DOPO aver eseguito le istruzioni all'interno del ciclo. Questo significa che il ciclo **do while** eseguirà le sue istruzioni almeno una volta, anche se la condizione è falsa la prima volta.

Il ciclo for

Il ciclo **for** si usa quando si sa in anticipo quante volte lo script deve essere eseguito.

```
for (inizio; test; incremento) {  
code to be executed;  
}
```

Parametri:

inizio: Inizializza il contatore del ciclo

test: Valuta ogni volta che il ciclo viene iterato, continuando se valuta **vero**, e terminando se valuta **falso**

increment: Aumenta il valore del contatore del ciclo

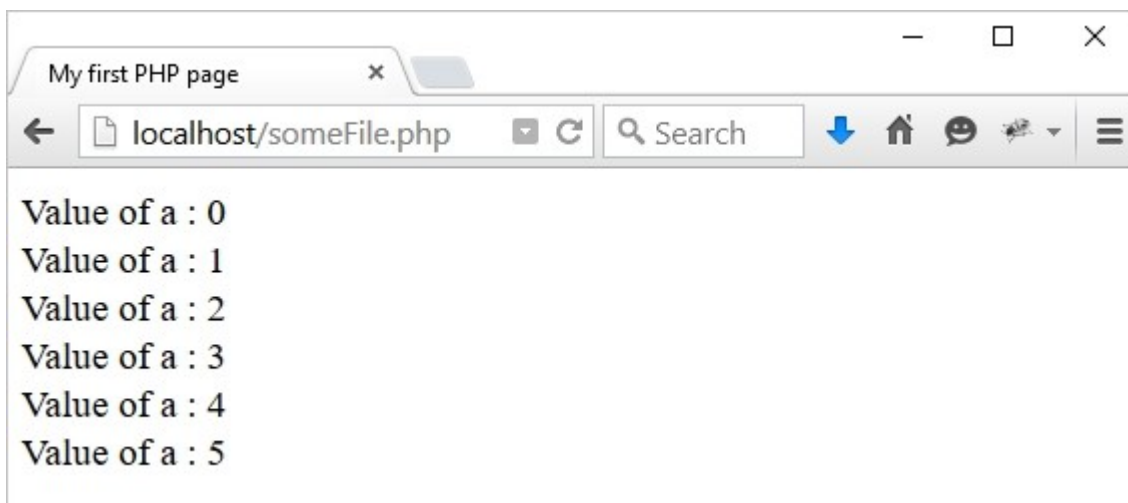
Ciascuna delle espressioni dei parametri può essere vuota o contenere più espressioni separate da Virgole.

Nell'istruzione **for**, i parametri sono separati da **punti e virgola**.

L'esempio qui sotto mostra i numeri da 0 a 5:

```
for ($a = 0; $a < 6; $a++) {  
echo "Value of a : ". $a . "<br />";  
}
```

Risultato:



Il ciclo **for** nell'esempio precedente prima imposta la variabile **\$a** a 0, poi controlla la condizione (**\$a < 6**). Se la condizione è vera, esegue il codice. Dopo di che, incrementa **\$a** (**\$a++**).

Il ciclo foreach

Il ciclo **foreach** funziona solo sugli array, ed è usato per fare un ciclo attraverso ogni coppia chiave/valore in un array. **Ci sono due sintassi:**

```
foreach ($array as $value) {  
code to be executed;  
}
```

```
//o foreach ($array as $key => $value) {  
code to be executed;  
}
```

La prima forma esegue un loop sull'array. Ad ogni iterazione, il valore dell'elemento corrente viene assegnato a **\$value** e il puntatore dell'array viene spostato di uno, fino a raggiungere l'ultimo elemento dell'array. La seconda forma assegnerà inoltre la chiave dell'elemento corrente alla variabile **\$key** ad ogni iterazione.

L'istruzione Switch

L'istruzione switch è un'alternativa all'if-elseif-else statement.

Utilizzare l'istruzione switch per selezionare uno dei vari blocchi di codice da eseguire.

Sintassi:

```
switch (n) {  
    case value1:  
        da eseguire se n=valore1  
    break;  
    case value2:  
        da eseguire se n=valore2  
    break;  
    ... default  
}
```

In primo luogo, la nostra singola espressione, **n** (il più delle volte una variabile), viene valutata una volta. Poi, il valore dell'espressione viene confrontato con il valore di ogni caso nella struttura. Se c'è una corrispondenza, viene eseguito il blocco di codice associato a quel caso.

Usando **dichiarazioni if else annidate** si ottiene un comportamento simile, ma switch offre una soluzione più elegante e ottimale.

Interruttore

Considerate il seguente esempio, che visualizza il messaggio appropriato per ogni giorno.

```
$today = 'Tue';

switch ($today) {
case "Mon":
echo "Today is Monday.";
break;
case "Tue":
echo "Today is Tuesday.";
break;
case "Wed":
echo "Today is Wednesday.";
break;
case "Thu":
echo "Today is Thursday.";
break;
case "Fri":
echo "Today is Friday.";
break;
case "Sat":
echo "Today is Saturday.";
break;
case "Sun":
echo "Today is Sunday.";
break;
default:
echo "Invalid day.";
}
//Outputs "Today is Tuesday."
```

La parola chiave **break** che segue ogni caso è usata per evitare che il codice passi automaticamente al caso successivo. Se si dimentica l'istruzione **break**, PHP continuerà automaticamente attraverso le istruzioni del caso successivo, anche quando il caso non corrisponde.

L'istruzione **predefinita** viene usata se non viene trovata alcuna corrispondenza.

```
$x=5;
switch ($x) {
case 1:
echo "One";
break;
case 2:
echo "Two";
break;
```

```
default: echo "Nessuna corrispondenza";  
}
```

```
//Output "Nessuna corrispondenza"
```

PHP

L'istruzione di **default** è opzionale, quindi può essere omessa.

Se non si specifica l'istruzione di **interruzione**, il PHP continua ad eseguire le istruzioni che seguono il **caso**, finché non trova un'**interruzione**. Puoi usare questo comportamento se hai bisogno di arrivare allo stesso output per più di un caso.

```
$day = 'Wed';
```

```
switch ($day) {  
  case 'Mon':  
    echo 'First day of the week';  
    break;  
  case 'Tue':  
  case 'Wed':  
  case 'Thu':  
    echo 'Working day';  
    break;  
  case 'Fri':  
    echo 'Friday!';  
    break;  
  default:  
    echo 'Weekend!';  
}
```

```
//Outputs "Working day"
```

L'esempio sopra avrà lo stesso risultato se **\$day** è uguale a 'Tue', 'Wed', o 'Thu'.

L'istruzione break

Come discusso nella lezione precedente, l'istruzione **break** è usata per uscire dallo **switch** quando un case è abbinato.

Se la pausa è assente, il codice continua a funzionare.

Per esempio:

```
$x=1;  
switch ($x) {  
case 1:  
echo "One";  
case 2:  
echo "Two";  
case 3:  
echo "Three";  
default:  
echo "No match";  
}
```

```
//Output "OneTwoThreeNo match"
```

Break può anche essere usato per fermare l'esecuzione di strutture **for**, **foreach**, **while**, **do-while**.

L'istruzione di interruzione termina l'attuale **for**, **foreach**, **while**, **do-while** o **switch** e continua ad eseguire il programma sulla linea che viene dopo il ciclo.

Un'istruzione di **interruzione** nella parte esterna di un programma (ad esempio, non in un ciclo di controllo) fermerà lo script.

L'istruzione continue

Quando viene usato all'interno di una struttura di looping, il **continue** permette di saltare ciò che rimane dell'iterazione corrente del ciclo. Quindi continua l'esecuzione alla valutazione della condizione e passa all'inizio dell'iterazione successiva.

L'esempio seguente salta i numeri pari nel ciclo **for**:

```
for ($i=0; $i<10; $i++) {  
  if ($i%2==0) {  
    continue ;  
  }  
  echo $i . ' ';  
}  
//Output: 1 3 5 7 9
```

Potete usare l'istruzione **continue** con tutte le strutture di looping.

L'istruzione include

Gli **include** e **require** permettono di inserire il contenuto di un file PHP in un altro file PHP, prima che il server lo esegua.

Includere i file fa risparmiare un bel po' di lavoro. Puoi creare un'intestazione, un piè di pagina o un file menu standard per tutte le tue pagine web. Poi, quando l'intestazione richiede un aggiornamento, puoi aggiornare solo il file di inclusione dell'intestazione.

Supponiamo di avere un file di intestazione standard chiamato **header.php**.

```
<?php  
echo '<h1>Welcome</h1>';  
?>
```

Usate l'istruzione **include** per includere il file di intestazione in una pagina.

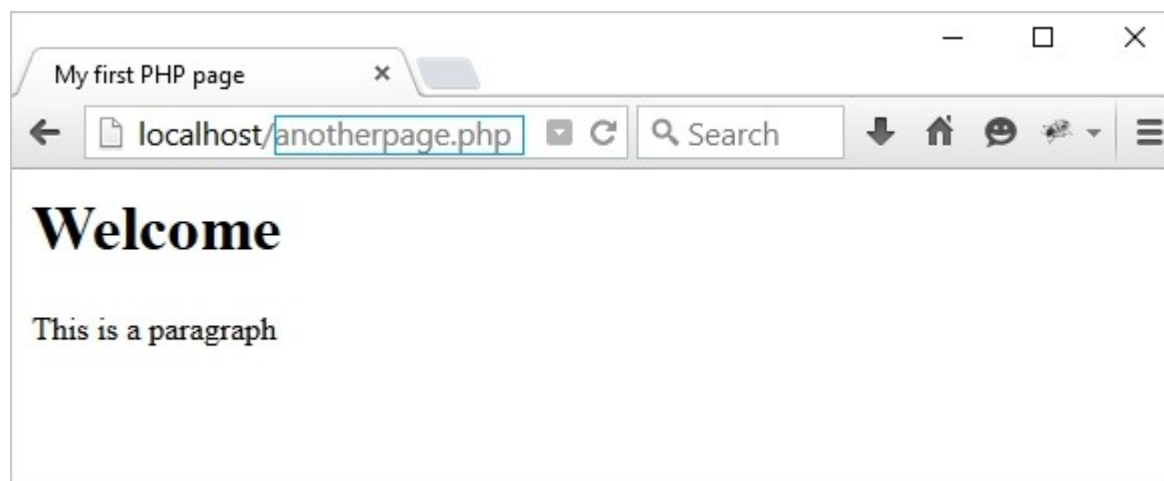
```
<html>  
<body> <?php include 'header.php'; ?>  
</html>
```

Le dichiarazioni include e require permettono di inserire il contenuto di un file PHP in un altro file PHP, prima che il server lo esegua.

Usando questo approccio, abbiamo la possibilità di includere lo stesso file **header.php** in più pagine.

```
<html>  
<body> <?php include 'header.php'; ?>  
</html>
```

Risultato:



I file sono inclusi in base al **percorso** del .

filePuoi usare un percorso **assoluto** o **relativo** per specificare quale file deve essere incluso.

include e require

Il **require** è identica a include, con l'eccezione che, in caso di fallimento, produce un errore fatale. Quando un file viene incluso usando l'**include** ma PHP non è in grado di trovarlo, lo script continua ad essere eseguito.

Nel caso del **require**, lo script cesserà l'esecuzione e produrrà un errore.

Utilizzare **require** quando il file è richiesto per l'esecuzione dell'applicazione.

Utilizzare **include** quando il file non è richiesto. L'applicazione dovrebbe continuare, anche quando il file non viene trovato.

Funzioni

Una **funzione** è un blocco di istruzioni che può essere usato ripetutamente in un programma.

Una funzione non verrà eseguita immediatamente al caricamento di una pagina. Sarà eseguita da una chiamata alla funzione.

Una dichiarazione di funzione definita dall'utente inizia con la parola **function**:

```
function functionName() {  
  //code to be executed  
}
```

Un nome di funzione può iniziare con una lettera o un trattino basso, ma non con un numero o un simbolo speciale.

I nomi delle funzioni NON sono sensibili alle maiuscole.

Nell'esempio qui sotto, creiamo la funzione **sayHello()**. La parentesi graffa di apertura ({} indica che questo è l'inizio del codice della funzione, mentre la parentesi graffa di chiusura (}) indica che questa è la fine.

Per chiamare la funzione, basta scrivere il suo nome:

```
function sayHello() {  
  echo "Hello!";  
}  
sayHello(); //Chiama la funzione
```

```
//Stampa "Ciao!"
```

Parametri di funzione

Gli argomenti sono specificati dopo il nome della funzione e tra le parentesi.

Le informazioni possono essere passate alle funzioni attraverso gli **argomenti**, che sono come variabili.

Gli argomenti sono specificati dopo il nome della funzione e tra le parentesi.

la nostra funzione prende un numero, lo moltiplica per due e stampa il risultato:

```
function multiplyByTwo($number) {  
    $answer = $number * 2;  
    echo $answer;  
}  
multiplyByTwo(3);  
//Stampa 6
```

Potete aggiungere tutti gli argomenti che volete, purché siano separati da **virgole**

```
function multiply($num1, $num2) {  
    echo $num1 * $num2;  
}  
multiply(3, 6);  
//Stampa 18
```

Quando si definisce una funzione, le variabili che rappresentano i valori che le verranno passati per l'elaborazione sono chiamate **parametri**. Tuttavia, quando usate una funzione, il valore che le passate è chiamato **argomento**.

Gli argomenti predefiniti possono essere definiti per gli argomenti della funzione.

Nell'esempio qui sotto, stiamo chiamando la funzione setCounter(). Non ci sono argomenti, quindi assumerà i valori predefiniti che sono stati definiti.

```
function setCounter($num=10) {  
    echo "Counter is ".$num;  
}  
setCounter(42); //Counter is 42  
setCounter(); //Counter è 10
```

Quando si usano argomenti predefiniti, qualsiasi default dovrebbe essere alla destra di qualsiasi argomento non predefinito; altrimenti, le cose non funzioneranno come previsto.

Return

Una funzione può restituire un valore usando il statement.

Return Return ferma l'esecuzione della funzione e rimanda il valore al codice chiamante. **Per**

esempio:

```
function mult($num1, $num2) {  
    $res = $num1 * $num2;  
    return $res;  
}  
echo mult(8, 3);  
//Stampa 24
```

Tralasciare return la funzione restituirà **NULL**

Una funzione non può restituire valori multipli, ma restituire un **array** produrrà risultati simili.

Variabili predefinite

È una variabile predefinita che è sempre accessibile, indipendentemente dallo scopo. È possibile accedere ai superglobal di PHP attraverso qualsiasi funzione, classe o file.

variabili superglobali di PHP sono `$_SERVER`, `$GLOBALS`, `$_REQUEST`, `$_POST`, `$_GET`, `$_FILES`, `$_ENV`, `$_COOKIE`, `$_SESSION`.

`$_SERVER`

`$_SERVER` è un array che include informazioni come intestazioni, percorsi e posizioni degli script. Le voci in questo array sono create dal server web. `$_SERVER['SCRIPT_NAME']` restituisce il percorso dello script corrente:

```
<?php
echo $_SERVER['SCRIPT_NAME'];
//Stampa "/somefile.php"
?>
```

Il nostro esempio è stato scritto in un file chiamato **somefile.php**, che si trova nella root del server web.

`$_SERVER['HTTP_HOST']` restituisce l'intestazione Host della richiesta corrente.

```
<?php
echo $_SERVER['HTTP_HOST'];
//Stampa "localhost"
?>
```

Questo metodo può essere utile quando hai molte immagini sul tuo server e hai bisogno di trasferire il sito web su un altro host. Invece di cambiare il percorso per ogni immagine, puoi fare come segue:

Crea un file **config.php**, che contiene il percorso delle tue immagini:

```
<?php
$host = $_SERVER['HTTP_HOST'];
$image_path = $host.'/images/';
?>
```

Usate il file **config.php** nei vostri script:

```
<?php
require 'config.php';
echo '<img src="'. $image_path. 'header.png' />';
?>
```

Il percorso delle tue immagini è ora dinamico. Cambierà automaticamente, in base all'intestazione dell'host.

Questo grafico mostra gli elementi principali di \$_SERVER.

Element/Code	Description
<code>\$_SERVER['PHP_SELF']</code>	Returns the filename of the currently executing script
<code>\$_SERVER['SERVER_ADDR']</code>	Returns the IP address of the host server
<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server
<code>\$_SERVER['HTTP_HOST']</code>	Returns the Host header from the current request
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page
<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the web server for communication (such as 80)
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	Returns the URI of the current page

`$_SERVER['HTTP_HOST']` restituisce l'intestazione Host della richiesta corrente.

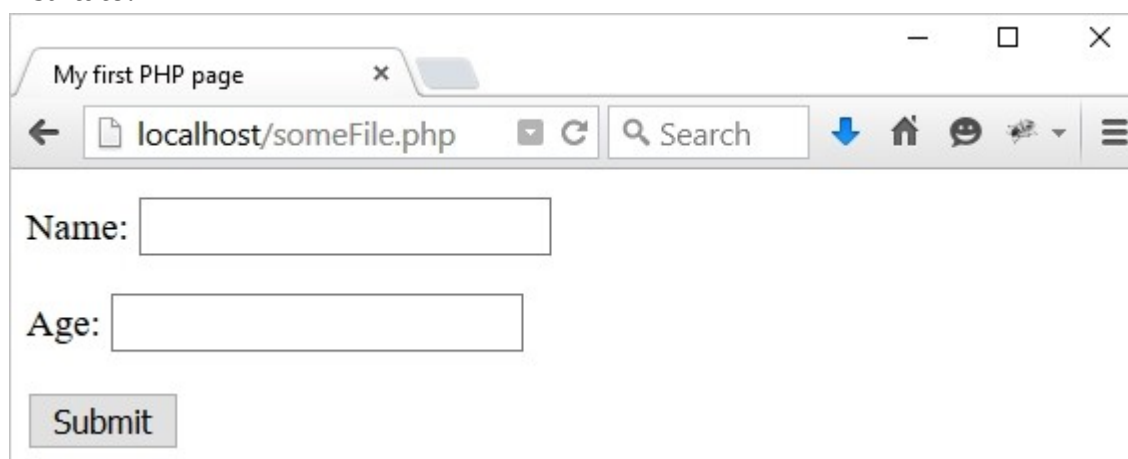
Form

Lo scopo dei superglobali PHP `$_GET` e `$_POST` raccolgono dati che sono stati inseriti in un modulo

L'esempio seguente mostra un semplice modulo HTML che include due campi di input e un pulsante di invio:

```
<form action=" first.php" method="post"> </form>
```

Risultato:



Lo scopo dei superglobali PHP `$_GET` e `$_POST` è di raccogliere i dati che sono stati inseriti in un modulo.

L'attributo **action** specifica che quando il modulo viene inviato, i dati vengono inviati a un file PHP chiamato **first.php**. L'attributo **method** sarà discusso più avanti. Per ora, imposteremo il valore a "post".

Ora, quando abbiamo un modulo HTML con l'attributo **action** impostato nel nostro file PHP, possiamo accedere ai dati del modulo inviato usando l'array associativo `$_POST`. Nel file **first.php**:

```
<html>
<body>
<form action=" first.php" method="post">
Benvenuto <?php echo $_POST["nome"]; ?><br />
La tua età: <?php echo $_POST["età"]?>
</form>
</html>
```

L'array superglobale `$_POST` contiene coppie chiave/valore. Nelle coppie, le chiavi sono i **nomi** dei controlli del modulo e i valori sono i **dati** inseriti dall'utente.

Abbiamo usato l'array `$_POST`, come **method="post"** was specified in the form. saperne di più sui metodi del modulo, premi **Continua!**

POST

I due metodi per l'invio di moduli sono GET e POST

Le **Informazioni** inviate da un modulo tramite il **POST** che è invisibile agli altri, poiché tutti i nomi e/o valori sono incorporati nel corpo della richiesta HTTP. Inoltre, non ci sono limiti alla quantità di informazioni da inviare. Inoltre, POST supporta funzionalità avanzate come il supporto per l'input binario in più parti durante il caricamento dei file sul server.

Tuttavia, non è possibile inserire un segnalibro nella pagina, poiché i valori inviati non sono visibili.

POST è il metodo preferito per inviare i dati del modulo.

GET

Le informazioni inviate tramite un modulo utilizzando il metodo **GET** sono visibili a tutti (tutti i nomi e i valori delle variabili sono visualizzati nell'**URL**). **GET** pone anche dei limiti alla quantità di informazioni che possono essere inviate - circa 2000 caratteri.

Tuttavia, poiché le variabili sono visualizzate nell'URL, è possibile inserire un segnalibro nella pagina, il che può essere utile in alcune situazioni.

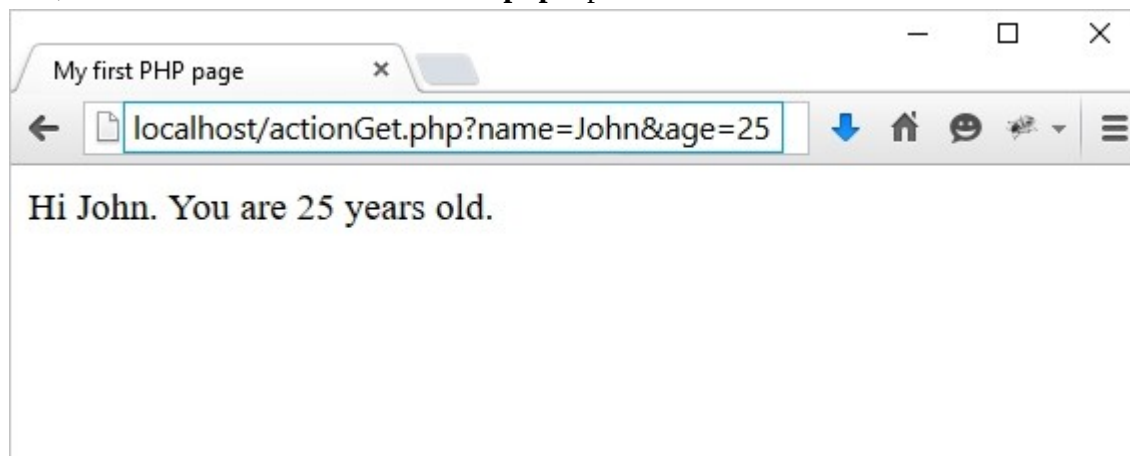
Per esempio:

```
<form action="actionGet.php" method="get">
  Name: <input type="text" name="name" /><br /><br />
  Age: <input type="text" name="age" /><br /><br />
  <input type="submit" name="submit" value="Submit" />
</form>
```

actionGet.php

```
<?php
  echo "Hi ".$_GET['name'].". ";
  echo "You are ".$_GET['age']."' years old.";
?>
```

Ora, il modulo è inviato all'**actionGet.php** e potete vedere i dati inviati nell'**URL**:



GET **non** dovrebbe **MAI** essere usato per inviare password o altre informazioni sensibili! **Quando si usa POST o GET, la corretta convalida dei dati del modulo attraverso il filtraggio e l'elaborazione è di vitale importanza per proteggere il tuo modulo dagli hacker e dagli exploit!**

Sessioni

Utilizzando una **sessione** è possibile memorizzare informazioni in variabili, da utilizzare in più pagine. Non vengono memorizzate sul computer dell'utente, come avviene con i **cookie** .
impostazione predefinita, le variabili di sessione durano fino a quando l'utente chiude il browser.

Avviare una sessione PHP

Una sessione viene avviata usando **session_start()** function.

Usate il globale PHP **\$_SESSION** per impostare le variabili di sessione.

```
<?php
// Start the session
session_start();

$_SESSION['color'] = "red";
$_SESSION['name'] = "John";
?>
```

Ora, le variabili di sessione **colore** e **nome** sono accessibili su più pagine, durante tutta la sessione. La funzione **session_start()** deve essere la prima cosa nel vostro documento. Prima di qualsiasi tag HTML.

Si può creare un'**altra pagina** che può accedere alle variabili di sessione che abbiamo impostato nella pagina precedente:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
echo "Your name is " . $_SESSION['name'];
// Outputs "Your name is John"
?>
</body>
</html>
```

Le vostre variabili di sessione rimangono disponibili in **\$_SESSION** superglobal fino alla chiusura della sessione.

Tutte le variabili globali di sessione possono essere rimosse manualmente usando **session_unset()**.

Si può anche distruggere la sessione con **session_destroy()**.

Cookie

I **cookie** sono spesso utilizzati per identificare l'utente. Un cookie è un piccolo file che il server incorpora nel computer dell'utente. Ogni volta che lo stesso computer richiede una pagina attraverso un browser, invierà anche il cookie. Con PHP, è possibile sia creare che recuperare i valori dei cookie. **Creare i cookie usando la funzione setcookie():**

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

name: Specifica il nome del cookie.

valore: `time()+86400*30`, imposterà il cookie a scadere tra 30 giorni. Se questo parametro viene omissso o impostato a 0, il cookie scadrà alla fine della sessione (quando il browser si chiude). Il valore predefinito è 0.

path: Specifica il percorso del server del cookie. Se impostato su `"/"`, il cookie sarà disponibile all'interno dell'intero dominio. Se impostato su `"/php/"`, il cookie sarà disponibile solo all'interno della directory php e di tutte le sottodirectory di php. Il valore predefinito è la directory corrente in cui viene impostato il cookie.

domain: Specifica il nome del dominio del cookie. Per rendere il cookie disponibile su tutti i sottodomini di `example.com`, impostare il dominio su `"example.com"`.

secure: Specifica se il cookie deve essere trasmesso solo su una connessione sicura HTTPS. TRUE indica che il cookie sarà impostato solo se esiste una connessione sicura. Il default è FALSE.

httponly: Se impostato su TRUE, il cookie sarà accessibile solo attraverso il protocollo HTTP (il cookie non sarà accessibile ai linguaggi di scripting). L'uso di httponly aiuta a ridurre il furto di identità tramite attacchi XSS. Il valore predefinito è FALSE.

Il parametro **name** è l'unico richiesto. Tutti gli altri parametri sono opzionali.

L'esempio seguente crea un cookie chiamato "user" con il valore "John". Il cookie scadrà dopo 30 giorni, che è scritto come $86.400 * 30$, in cui 86.400 secondi = un giorno. La '/' significa che il cookie è disponibile in tutto il sito web.

Poi recuperiamo il valore del cookie "utente" (usando la variabile globale `$_COOKIE`). Usiamo anche la funzione **isset()** per scoprire se il cookie è impostato:

```
<?php
$value = "John";
setcookie("user", $value, time() + (86400 * 30), '/');

if(isset($_COOKIE['user'])) {
    echo "Value is: ". $_COOKIE['user'];
}
//Stampa "Value is: John"
?>
```

Il **setcookie()** **DEVE apparire prima del tag** `<html>`.

valore del cookie viene automaticamente codificato quando il cookie viene inviato, e viene automaticamente decodificato quando viene ricevuto. Tuttavia, **non** memorizzate **MAI** informazioni sensibili nei cookie.

Manipolare i file

PHP offre un certo numero di funzioni da usare per creare, leggere, caricare e modificare i file.s. La funzione **fopen()** crea o apre un file. Se usate **fopen()** with a file that does not exist, the file will be created, given that the file has been opened for writing (w) or appending (a).

usate uno dei seguenti **modi** per aprire il file.

r: Apre il file in sola lettura.

w: Apre il file in sola scrittura. Cancella il contenuto del file o crea un nuovo file se non esiste.

a: Apre il file per la sola scrittura.

x: Crea un nuovo file in sola scrittura.

r+: Apre il file in lettura/scrittura.

w+: Apre il file in lettura/scrittura. Cancella il contenuto del file o crea un nuovo file se non esiste.

a+: Apre il file in lettura/scrittura. Crea un nuovo file se il file non esiste

x+: Crea un nuovo file in lettura/scrittura.

esempio seguente crea un nuovo file, "file.txt", che sarà creato nella stessa directory che ospita il codice PHP.

```
$myfile = fopen("file.txt", "w");
```

PHP offre un certo numero di funzioni da usare per creare, leggere, caricare e modificare i file.

Scrivere su file

Quando si scrive su un file, si usa **fwrite()** function.

Il primo parametro di **fwrite()** is the file to write to; the second parameter is the string to be written.

L'esempio seguente scrive una coppia di nomi in un nuovo file chiamato "nomi.txt".

```
<?php
    $myfile = fopen("names.txt", "w");
    $txt = "John\n";
    fwrite($myfile, $txt);
    $txt = "David\n";
    fwrite($myfile, $txt);
    fclose($myfile);
    /* il file conterrà:
    John
    David */
?>
```

Notate che abbiamo scritto al file "names.txt" due volte, e poi abbiamo usato la **funzione** fclose() per chiudere il file.

Il simbolo \n si usa quando si scrivono **nuove linee**.

fclose()

La funzione **fclose()** chiude un file aperto e restituisce TRUE al successo o FALSE al fallimento. È una buona pratica chiudere tutti i file dopo aver finito di lavorarci.

Aggiunta a un file

Se volete aggiungere del contenuto a un file, dovete aprire il file in **modalità append**. Per esempio:

```
$myFile = "test.txt";  
$fh = fopen($myFile, 'a');  
fwrite($fh, "Some text");  
fclose($fh);
```

Quando si aggiunge a un file usando la **modalità 'a'**, il puntatore **del file** è posto alla fine del file, assicurando che tutti i nuovi dati siano aggiunti alla fine del file.

Creiamo un esempio di un modulo che aggiunge i dati compilati a un file.

```
<?php  
if(isset($_POST['text'])) {  
    $name = $_POST['text'];  
    $handle = fopen('names.txt', 'a');  
    fwrite($handle, $name."\n");  
    fclose($handle);  
}  
?>  
  
<form method="post">  
Name: <input type="text" name="text" />  
<input type="submit" name="submit" />  
</form>
```

PHP

Ora, ogni volta che un nome viene inserito e inviato, viene aggiunto al file "names.txt", insieme a una nuova riga.

La funzione **isset()** determinava se il modulo era stato inviato, così come se il testo conteneva un valore.

Non abbiamo specificato un attributo di **azione** per il modulo, quindi si invierà da solo.

Leggere un file

La funzione **file()** legge l'intero file in un array. Ogni elemento della matrice corrisponde a una riga del file:

```
<?php
$myfile = fopen("names.txt", "w");
$txt = "John\n";
fwrite($myfile, $txt);
$txt = "David\n";
fwrite($myfile, $txt);
fclose($myfile);
$read = file('names.txt');
foreach ($read as $line) {
echo $line . ", ";
}
?>
```

Questo stampa tutte le linee del file e le separa con delle virgole.

Abbiamo usato il ciclo **foreach**, perché la variabile **\$read** è un **array**.

Alla fine dell'output nell'esempio precedente, avremmo una virgola, poiché la stampiamo dopo ogni elemento dell'array.

Il codice seguente ci permette di evitare di stampare la virgola finale.

```
<?php
$myfile = fopen("names.txt", "w");

$txt = "John\n";
fwrite($myfile, $txt);
$txt = "David\n";
fwrite($myfile, $txt);

fclose($myfile);

$read = file('names.txt');
foreach ($read as $line) {
echo $line . ", ";
}
?>
```

La variabile **\$count** usa il **count()** function to obtain the number of elements in the **\$read** array. Poi, nel ciclo **foreach**, dopo ogni riga stampata, determiniamo se la riga corrente è inferiore al numero totale di righe, e stampiamo una virgola se lo è.

Questo evita di stampare la virgola finale, poiché per l'ultima riga, **\$i** è uguale a **\$count**.

Classi e oggetti in PHP

La programmazione orientata agli oggetti (OOP) è uno stile di programmazione che ha lo scopo di rendere il pensiero della programmazione più vicino al pensiero del mondo reale. Gli **oggetti** sono creati usando le **classi**, che sono il punto focale di OOP.

Una classe descrive ciò che l'oggetto sarà, ma è separata dall'oggetto stesso. In altre parole, una classe può essere pensata come il progetto di un oggetto, descrizione o definizione.



Qui, **Building** è una class. Definisce le caratteristiche di un edificio generico e come dovrebbe funzionare. L'**Empire State Building** è un oggetto specifico (o **istanza**) di quella classe. Potete usare la stessa classe come modello per creare più oggetti diversi.

Classi PHP

In PHP, una classe può includere variabili membro chiamate **proprietà** per definire le caratteristiche di un oggetto, e funzioni, chiamate **metodi**, per definire il comportamento di un oggetto. La definizione di una classe inizia con la parola chiave **class**, seguita da un nome di classe. Le parentesi graffe racchiudono le definizioni delle proprietà e dei metodi appartenenti alla classe. **Per esempio:**

```
class Person {  
    public $age; //property  
    public function speak() { //method  
        echo "Hi!";  
    }  
}
```

Il codice qui sopra definisce una classe **Person** che include una proprietà **age** e un metodo **speak()**. Un nome di classe valido inizia con una lettera o un trattino basso, seguito da un numero qualsiasi di lettere, numeri o trattini bassi.

Notate la parola chiave **public** davanti al metodo **speak**; è uno **specificatore di visibilità**. La parola chiave **public** specifica che il membro è accessibile da qualsiasi punto del codice.

Ci sono altre parole chiave di visibilità e le impareremo nelle lezioni successive.

Controlla la prossima lezione per vedere come **istanziare gli oggetti**!

Oggetti PHP

Il processo di creazione di un oggetto di una classe è chiamato **istanziare** .

Per istanziare un oggetto di una classe, usate la parola chiave **new**, come nell'esempio qui sotto:

```
$bob = new Person();
```

Nel codice qui sopra, **\$bob** è un oggetto della **persona** class.

Per accedere alle proprietà e ai metodi di un oggetto, usate il costrutto freccia (->), come in:

```
echo $bob->age;
```

Questa dichiarazione produce il valore della proprietà **age** . Se volete assegnare un valore a una proprietà usate l'operatore di assegnazione = come fareste con qualsiasi variabile.

Definire la **classe Person**, istanziare un oggetto, fare un'assegnazione e chiamare il metodo **speak()**:

```
<?php
class Person {
public $age;
function speak() {
echo "Hi!";
}
}
$p1 = new Person();
$p1->age = 23;
echo $p1->age;
$p1->speak();
?>
```

\$this

\$this è una pseudo-variabile che è un riferimento all'oggetto chiamante. Quando lavorate all'interno di un metodo, usate **\$this** nello stesso modo in cui usereste il nome di un oggetto fuori dalla classe.

Per esempio:

```
<?php
class Dog {
public $legs=4;
public function display() {
echo $this->legs;
}
}
$d1 = new Dog();
$d1->display();

echo '<br />';

$d2 = new Dog();
$d2->legs = 2;
$d2->display();
?>
```

Abbiamo creato due oggetti della classe Dog e chiamato i loro metodi **display()**. Poiché il metodo display() usa **\$this**, il valore delle **gambe** faceva riferimento al valore della proprietà dell'oggetto chiamante. Come potete vedere, ogni oggetto può avere i propri valori per le proprietà della classe.

Costruttore di classi PHP

PHP fornisce il metodo magico costruttore **__construct()**, che viene chiamato automaticamente ogni volta che viene istanziato un nuovo oggetto.

Per esempio:

```
<?php
class Person {
public function __construct() {
echo "Object created";
}
}
$p = new Person();
?>
```

Il metodo **__construct()** è spesso usato per qualsiasi inizializzazione di cui l'oggetto può aver bisogno prima di essere usato. I parametri possono essere inclusi in **__construct()** per accettare valori quando l'oggetto viene creato.

Per esempio:

```
<?php
class Person {
public $name;
public $age;
public function __construct($name, $age) {
$this->name = $name;
$this->age = $age;
}
}
$p = new Person("David", 42);
echo $p->name;
?>
```

Nel codice sopra, il costruttore usa gli argomenti nell'istruzione `new` per inizializzare le proprietà di classe corrispondenti.

Non si possono scrivere più metodi **__construct()** con diversi numeri di parametri. Comportamenti diversi del costruttore devono essere gestiti con la logica all'interno di un singolo metodo **__construct()**.

Distruttore di classe PHP

Simile al costruttore di classe, c'è un metodo magico destructor **__destruct()**, che viene chiamato automaticamente quando un oggetto viene distrutto.

Per esempio:

```
<?php
class Person {
public function __destruct() {
echo "Object destroyed";
}
}
$p = new Person();
?>
```

Questo script crea un nuovo oggetto Persona. Quando lo script finisce, l'oggetto viene automaticamente distrutto, il che chiama il distruttore e produce il messaggio "Object destroyed". Per attivare esplicitamente il distruttore, potete distruggere l'oggetto usando la funzione **unset()** in una dichiarazione simile a:

unset(\$p);

I distruttori sono utili per eseguire certi compiti quando l'oggetto finisce il suo ciclo di vita. Per esempio, rilasciare risorse, scrivere file di log, chiudere una connessione al database, e così via. PHP rilascia tutte le risorse quando uno script termina la sua esecuzione.

Ereditarietà delle classi PHP

Le classi possono ereditare i metodi e le proprietà di un'altra classe. La classe che eredita i metodi e le proprietà è chiamata **sottoclasse**. La classe da cui una sottoclasse eredita è chiamata **classe genitore**. L'ereditarietà .

si ottiene usando la parola chiave **extends**. Per esempio:

```
<?php
class Animal {
    public $name;
    public function hi() {
        echo "Hi from Animal";
    }
}
class Dog extends Animal {
}

$d = new Dog();
$d->hi();

?>
```

Qui la classe **Dog** eredita dalla classe **Animal**. Come potete vedere, tutte le proprietà e i metodi di **Animal** sono accessibili a **Dog** objects.

I costruttori genitori non sono chiamati implicitamente se la sottoclasse definisce un costruttore. Tuttavia, se il figlio non definisce un costruttore, allora sarà ereditato dalla classe padre se non è dichiarato **privato**.

Notate che tutte le nostre proprietà e i nostri metodi sono **public**.

Per un maggiore controllo sugli oggetti, dichiarate metodi e proprietà usando una parola chiave di visibilità. Questo controlla come e da dove si può accedere a proprietà e metodi.

Controllate la prossima lezione per saperne di più sulla **visibilità**.

Visibilità PHP

Rende i membri accessibili solo dalla classe che li definisce.

La **visibilità** controlla come e da dove si effettua l'accesso di **proprietà** e **metodi**.

Finora, abbiamo usato il **public** per specificare che una proprietà/metodo è accessibile da qualsiasi luogo.

Ci sono altre due parole chiave per dichiarare la visibilità:

protected: Rende i membri accessibili solo all'interno della classe stessa, attraverso l'ereditarietà, e dalle classi madri.

private: Rende i membri accessibili solo dalla classe che li definisce.

Le proprietà della classe devono sempre avere un tipo di visibilità. I metodi dichiarati senza alcuna parola chiave di visibilità esplicita sono definiti come **public**.

I membri **protected** sono usati con l'ereditarietà.

I membri **private** sono usati solo internamente in una classe.

Interfacce PHP

Un **interfaccia** specifica una lista di metodi che una classe **deve** implementare. Tuttavia, l'interfaccia stessa non contiene alcuna implementazione di metodi. Questo è un aspetto importante delle interfacce perché permette ad un metodo di essere gestito in modo diverso in ogni classe che usa l'interfaccia.

L'**interface** definisce l'interfaccia.

L'**implements** è usata in una classe per implementare un'interfaccia.

Per esempio, **AnimalInterface** è definita con una dichiarazione per la funzione **makeSound()**, ma non è implementata finché non è usata in una classe:

```
<?php
interface AnimalInterface {
    public function makeSound();
}

class Dog implements AnimalInterface {
    public function makeSound() {
        echo "Woof! <br />";
    }
}

class Cat implements AnimalInterface {
    public function makeSound() {
        echo "Meow! <br />";
    }
}

$myObj1 = new Dog();
$myObj1->makeSound();

$myObj2 = new Cat();
$myObj2->makeSound();
?>
```

Una classe può implementare più interfacce. Più interfacce possono essere specificate separandole con delle virgole.

Per esempio:

```
class Demo implements AInterface, BInterface, CInterface {
    // Functions declared in interfaces must be defined here
}
```

Un'interfaccia può essere ereditata da un'altra interfaccia usando la parola chiave **extends**.

Tutti i metodi specificati in un'interfaccia richiedono visibilità **pubblica**.

Classi astratte PHP

Le classi astratte possono essere ereditate ma non possono essere istanziate.

Offrono il vantaggio di poter contenere sia metodi con definizioni che metodi astratti che non sono definiti finché non vengono ereditati. Una classe che eredita da una classe astratta deve implementare tutti i metodi astratti.

La parola chiave **abstract** è usata per creare una classe astratta o un metodo astratto.

Per esempio:

```
<?php
abstract class Fruit {
    private $color;
    abstract public function eat();
    public function setColor($c) {
        $this->color = $c;
    }
}

class Apple extends Fruit {
    public function eat() {
        echo "Omnomnom";
    }
}

$obj = new Apple();
$obj->eat();

?>
```

Le funzioni astratte possono apparire solo in una classe astratta.

La parola chiave static

In PHP **static** è Una parola chiave definisce proprietà e metodi statici.

Si può accedere a una proprietà/metodo statico di una classe senza creare un oggetto di quella classe.

proprietà o un metodo statico è accessibile usando l'**operatore di risoluzione dello scopo ::** tra il nome della classe e il nome della proprietà/metodo.

Per esempio:

```
<?php
class myClass {
    static $myStaticProperty = 42;
}

echo myClass::$myStaticProperty;
?>
```

La parola chiave **self** è necessaria per accedere a una proprietà statica da un metodo statico in una definizione di classe. **Per esempio:**

```
<?php
class myClass {
    static $myStaticProperty = 42;
}

echo myClass::$myStaticProperty;
?>
```

Gli oggetti di una classe non possono accedere alle proprietà statiche della classe, ma possono accedere ai metodi statici:

```
<?php
class myClass {
    static $myProperty = 42;
    static function myMethod() {
        echo self::$myProperty;
    }
}

myClass::myMethod();
?>
```

La parola chiave final

La parola chiave PHP final definisce i metodi che non possono essere sovrascritti nelle classi figlio. Le classi che sono definite finali non possono essere ereditate.

Questo esempio dimostra che un metodo finale non può essere sovrascritto in una classe figlia:

```
<?php
class myClass {
    final function myFunction() {
        echo "Parent";
    }
}

class myClass2 extends myClass {
    function myFunction() {
        echo "Child";
    }
}

// ERRORE perché un metodo finale non può essere sovrascritto
?>
```

Il seguente codice dimostra che una classe finale non può essere ereditata:

```
<?php
final class myFinalClass {
}
class myClass extends myFinalClass {
}
// ERRORE perché una classe finale non può essere ereditata.

?>
```

A differenza delle classi e dei metodi, le proprietà non possono essere marcate come **final**.