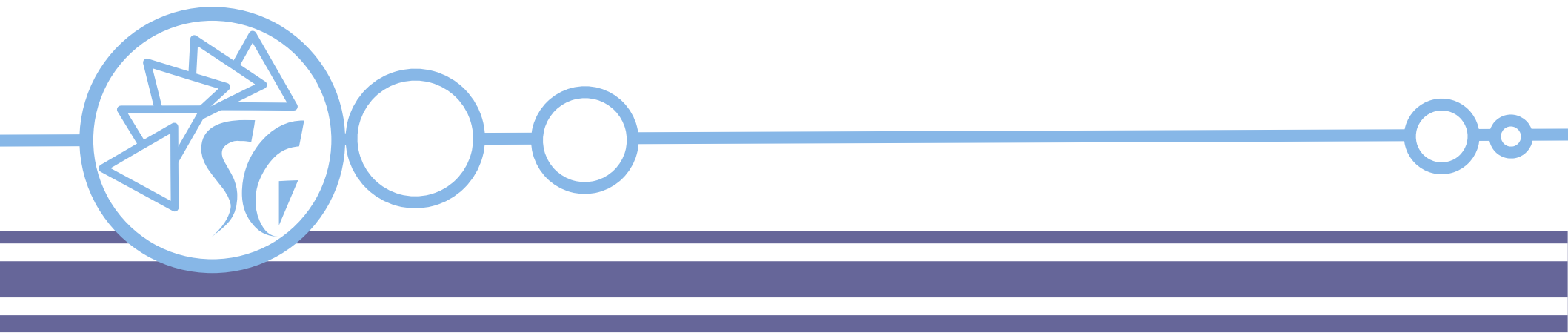

PHP

Ing. Simone Giustetti

www.giustetti.net

PHP

Linguaggio di programmazione orientato al web.

Il nome ufficiale è **PHP: Hypertext Preprocessor**.

Progettazione e sviluppo incominciano nel 1994.
PHP 3 rilasciato nel 1998. Supporto base per gli oggetti.
PHP 4 rilasciato nel 2000.
PHP 5 rilasciato nel 2008.
PHP 7 rilasciato nel 2015.
PHP 8 rilasciato nel 2020. Ultima versione stabile.

È un **linguaggio interpretato**, eseguito al volo dall'interprete, che deve essere presente sulla macchina.

È un **linguaggio imperativo**: Il programmatore deve scrivere codice dettagliato per tutte le operazioni eseguite dall'interprete.

È un **linguaggio general purpose**: Consente di sviluppare siti web, applicazioni a riga di comando e applicazioni grafiche tramite le librerie **PHPGtk** o **wxWidgets**.

Supporta sia il paradigma di programmazione **imperativo strutturato**, che **la programmazione ad oggetti**.

È un linguaggio a tipizzazione debole. La tipologia dei dati può essere cambiata arbitrariamente dallo interprete in base alla operazioni che esegue.

Il formato “predefinito” è: **stringa di testo**.
Bisogna fare attenzione quando si eseguono operazioni matematiche / logiche sui dati.

PEAR e PECL

PHP Extension & **A**pplication **R**epository è la libreria standard di PHP, che include estensioni e funzioni addizionali per il linguaggio.

PEAR è suddiviso in moduli sviluppati in maniera indipendente, che possono essere installati con un apposito programma.

PECL è un insieme di estensioni di PHP scritte nel linguaggio **C**. Possono essere installate con il comando **pecl**, oppure devono essere compilate a partire dal codice sorgente.

Dichiarare Codice PHP

Qualsiasi file di testo che contenga istruzioni racchiuse tra i tag **<?php** ... **?>** è un programma.

I tag possono essere mescolati con quelli del linguaggio HTML, ma devono sempre essere in coppia.

È possibile inserire più coppie di tag in una singola pagina HTML.

L'interprete esegue il codice sito tra le coppie di tag.

Dichiarare Codice PHP – Esempi

```php
<?php
   echo( 'Ciao Mondo !' );
?>

<!DOCTYPE html>
<html>
   <head>
      <title>La mia prima pagina PHP</title>
      <meta charset="utf-8" />
   </head>
   <body>
      <?php
         echo( 'Ciao Mondo !' . PHP_EOL );
      ?>
   </body>
</html>
```

I Commenti

Esistono due tipologie di commenti in PHP:

- I commenti di una singola riga

Ogni riga che incomincia con due slash (Shift+7).

```
Codice...  // Commento
// Commento
```

- I commenti inclusi in un blocco di testo

```
Codice...  /* Commento
Commento
Commento */  Codice...
```

Separare / Raggruppare le Istruzioni

Le istruzioni devono essere terminate dal simbolo “**;**” (**Punto e virgola**). Più istruzioni possono essere scritte sulla medesima riga di testo.

Le istruzioni possono essere raggruppate inglobandole tra parentesi graffe (**{** … **}**). I blocchi di istruzioni possono essere scritti su più righe.

L’uso dell’indentazione è opzionale, ma fortemente consigliato per migliorare la leggibilità del codice.

Le Variabili

Una variabile è una porzione di memoria in cui vengono salvate informazioni per la durata in esecuzione di uno script.

Le variabili devono essere dichiarate mediante assegnazione di un valore (implicitamente) prima dell'uso.

```
$variabile = "test";  // Dichiarazione implicita
$variabile = 5;       // Dichiarazione implicita
$variabile;           // Segnala un errore
```

Tipologie di Dato

PHP riconosce le seguenti tipologie di dato:

- bool
- int
- float (double)
- stringhe di testo
- array
- oggetti
- callable
- iterable
- resource
- NULL

Visibilità (Scope) delle Variabili

La visibilità di una variabile è limitata al blocco di istruzioni in cui viene dichiarata. Nella maggior parte dei casi lo scope coincide con il file in cui è dichiarata e quelli inclusi di seguito.

Le variabili definite in una funzione sono visibili solo all'interno della funzione stessa.

L'istruzione **global** impone l'uso di una variabile globale quando potrebbe sussistere ambiguità.

Variabili Esterne

PHP prevede alcune variabili **sempre accessibili** attraverso cui leggere i dettagli di una richiesta HTTP, l'ambiente di uno script, ecc.:

- $_COOKIE
- $_ENV: Variabili di ambiente.
- $_FILES: Elenco dei file in upload.
- $_GET
- $_POST
- $_REQUEST: $_GET ∪ $_POST ∪ $_COOKIE
- $_SERVER
- $_SESSION: Variabili di sessione.
- $GLOBALS: Elenco variabili ∈ scope globale.

Le Costanti

Una costante è una porzione di memoria in cui vengono salvate informazioni che non possono essere successivamente modificate.

Le costanti devono essere dichiarate con l'istruzione **define("**<const>**",** <value> **);**.

È prassi battezzare le costanti con lettere solo maiuscole.

Le costanti hanno visibilità globale.

Gli Array

Un array è una variabile che contiene molti dati dello stesso tipo.

In PHP gli array sono **associativi**. Gli identificatori delle componenti non sono limitati a numeri interi, è possibile usare stringhe descrittive.

Per accedere ad un elemento di un array è necessario il ricorso alle parentesi quadre.

Ar_01[‘3’], Ar_01[‘colore’], ...

Le componenti numeriche partono da **0**.

Gli Oggetti

Strutture dati composte da variabili e codice (Membri e Metodi).

Per dichiarare un **oggetto** è necessario prima definire una **classe**.

Per dichiarare un **oggetto** si fa ricorso alla istruzione **new**.

Per accedere ad un elemento è necessario il ricorso all'operatore **->** (Freccia):

```
Obj_01->cognome
```

Gli Operatori

Gli operatori eseguono operazioni elementari sui dati.

Il risultato di un'operazione può variare in base alla tipologia del dato.

Esistono operatori specifici per ogni tipologia di dato.

Alcuni operatori hanno la precedenza su altri. Usare le parentesi tonde per imporre la precedenza tra le operazioni.

Gli Operatori di Assegnazione

Assegnano un valore ad una variabile.

- **=** (Assegnazione per valore)
- **= &** (Assegnazione per reference)

L’assegnazione per reference implica che non vi sia alcuna copia. Entrambe le variabili puntano al medesimo dato.

Gli Operatori Matematici

Operano sui numeri e rendono sempre un valore numerico.

- **+**, **-** (Addizione e sottrazione algebrica)
- ***, */* (Moltiplicazione e divisione algebrica)
- **%** Il resto di una divisione intera (modulo)
- **++** Incremento (+ 1)
- **--** Decremento (- 1)
- ** Elevazione a potenza
- **+=**, **-=**, ***=**, **/=**, **%=**, ****=** Eseguono un'operazione ed assegnano il risultato

Gli Operatori Stringa

Operano su stringhe di testo rendono sempre una stringa.

- **.** (Concatenazione)
- **.=** Esegue la concatenazione ed assegna il risultato

Gli Operatori sui Bit

Gli operatori booleani sono usati per eseguire operazioni sui bit di un intero.

- **>>** (Shift a destra)
- **<<** (Shift a sinistra)
- **&** (And)
- **|** (Or)
- **~** (Not)
- **^** (Xor)

Gli Operatori Logici

Gli operatori logici sono usati per eseguire confronti e rendono sempre un valore **FALSE** oppure **TRUE**.

- **&&**, **and** (Vero se entrambe le espressioni sono vere)
- **||**, **or** (Vero quando almeno una espressione lo è)
- **xor** (Vero quando entrambe le espressioni hanno medesimo valore)
- **!** (Not. Nega un’espressione)

Gli Operatori di Confronto

Gli operatori di confronto sono usati per confrontare valori e tipologie di dato.

- **==** (Valore uguale)
- **<>**, **!=** (Valore diverso)
- **===** (Valore e tipologia di dato uguale)
- **!==** (Valore o tipologia di dato diversi)
- **>**, **>=** (Maggiore / Maggiore o Uguale)
- **<**, **<=** (Minore / Minore o Uguale)
- **<=>** (Rende un intero minore, uguale o maggiore di zero in base all'esito del confronto)

Gli Operatori per Array

Alcuni operatori visti precedentemente possono essere impiegati con gli array, ma assumono un comportamento particolare.

- **+** (Unione di 2 array)
- **==** (Medesime coppie chiave / valore)
- **<>** / **!=** (Gli array sono diversi)
- **===** (Medesime coppie chiave / valore nel medesimo ordine e del medesimo tipo)
- **!==** (Gli array sono diversi, l'ordine delle chiavi o la tipologia dei valori differiscono)

Gli Operatori di Tipologia

L’operatore **instanceof** può essere usato per verificare la tipologia di una variabile.

```
// Rende "TRUE" o "FALSE"
$a instanceof classe_01
```

Eliminare una Variabile

Una variabile può essere "pulita" utilizzando la funzione **unset()**:

```
unset( $var );  // Variabile ora inutilizzabile
```

Utilizzando la funzione **unset()** si possono cancellare anche singole componenti di un array:

```
unset( $arr[ ‘5’ ]);  // Rimuove l’elemento 5
unset( $arr );        // Rimuove l’intero array
```

Ciclo if

Un blocco di codice è eseguito solo se è verificata una condizione.

```
if( <condizione> )  {
   <Blocco 1 di istruzioni>
   ...
}  else  {
   <Blocco 2 di istruzioni>
   ...
}
```

Il blocco **else** è opzionale.

Ciclo if

È possibile annidare le istruzioni **if**, oppure usare il costrutto **elseif**, quando bisogna verificare molte condizioni in cascata.

```
if( <condizione 1> )  {
   <Blocco 1 di istruzioni>
}  elseif( <condizione 2> )  {
   <Blocco 2 di istruzioni>
}  else  {
   <Blocco 3 di istruzioni>
}
```

Ciclo switch

Costrutto condizionale che permette di verificare tutti i valori resi da una espressione.

```
switch( <espressione> ) {
    case <val01>:
      <Blocco 1 di istruzioni>
    break;
    case <val02>:
      <Blocco 2 di istruzioni>
    break;
    ...
    default:
      <Blocco n di istruzioni>
}
```

Ciclo match

Costrutto condizionale, simile a switch, che assegna un valore tra molti.

```
<variabile> = match( <espressione> )  {
    <val01> => <espressione01>,
    <val02>, <val03> => <espressione02>,
    ...
    default => <espressione0N>,
};
```

Equivale all'operatore di confronto **===**

<espressioneXX> può essere un valore, oppure una funzione che renda un valore.

Operatore Condizionale Ternario

L’operatore condizionale ternario lavora con tre valori. La sintassi è:

<condizione> **?** <espr01> **:** <espr02> **;**

È una sintassi compatta per un ciclo if, then, else.

Es:

```
anni >= 18 ? "maggiorenne" : "minorenne";
```

Ciclo while

Esegue un blocco di codice un numero molteplice di volte in base alla verifica di una condizione.

```
while( <condizione> )  {
   <Blocco 1 di istruzioni>;
}
```

È possibile che il blocco non sia mai eseguito

È possibile che il blocco sia eseguito all'infinito, se la condizione di stop non fosse mai verificata.

Ciclo do..while

La verifica della condizione di esecuzione è posta in coda al ciclo invece che in testa.

```
do  {
   <Blocco di istruzioni>;
}  while( <condizione> );
```

Il ciclo viene sempre eseguito almeno una volta.

Ciclo for

I cicli **for** usano un contatore che viene incrementato ad ogni iterazione.

```
for( <condizione_iniziale>; <condizione_finale>;
   <incremento> )  {
   <Blocco di istruzioni>;
}
```

Viene eseguito sempre un numero determinato di volte.

Iterare gli Elementi di un Array

In PHP esiste l'istruzione **foreach** che opera sul contenuto di un array iterando tra le componenti.

```
foreach( <array> as <$value> )  {
   <Blocco di istruzioni>;
}

foreach( <array> as <$key> => <$value> )  {
   <Blocco di istruzioni>;
}
```

Eseguito un numero di volte pari alle componenti.

Cambiare il Flusso di un Ciclo

Le istruzioni **break** e **continue** cambiano il normale flusso di esecuzione di un ciclo.

- break: Causa l'uscita da un ciclo spostando il flusso del programma alla prima riga successiva il ciclo stesso.
- continue: Interrompe l'iterazione in corso e riprende il ciclo dall'iterazione successiva. Consente di saltare iterazioni.

Includere Script PHP in Altri

Le istruzioni **include** e **require** importano le istruzioni di un altro script all'interno del chiamante.

- include: Causa la lettura e l'esecuzione delle istruzioni contenute in un file diverso dal chiamante.
- require: Opera come include, ma interrompe l'esecuzione dello script e rende un errore se non esistesse il file da importare.

Le istruzioni **include_once** e **require_once** aprono uno script esterno una volta sola.

Funzioni

Le funzioni in PHP devono essere dichiarate con l'istruzione **function**.

```
function <nome>( $<parametro01>, … )  {
    …;
}
```

Il corpo di una funzione può contenere qualsiasi istruzione PHP valida e sintatticamente corretta.

Le funzioni devono essere dichiarate prima del loro uso.

Funzioni

Le funzioni hanno **scope globale**.

PHP non supporta lo overloading delle funzioni.

Le funzioni non possono essere cancellate o ridefinite.

Le funzioni possono rendere un valore mediante l'istruzione **return** nel corpo della funzione.

```
return $var01;
```

Parametri di Funzione

I parametri di una funzione sono passati per valore. Se il valore venisse modificato all'interno della funzione, le modifiche andrebbero perse all'uscita dalla stessa.

È consentito passare i parametri per reference

```
function <nome>( & $<parametro01>, … )  {
    …;
}
```

Le modifiche apportate permangono all'uscita.

Parametri di Default

È possibile assegnare un valore predefinito ai parametri. Quando la chiamata alla funzione non include il parametro, viene usato il valore predefinito.

```
function <nome>( $<param01>,
                 ... ,
                 $<param02> = <valore> )  {
   ...;
}
```

Parametri di Default

Quando viene passato un valore per un parametro con default, il valore viene usato al posto del default. Ciò vale anche per il valore **null**.

I parametri di default devono essere elencati sempre per ultimi.

I valori di default devono essere delle costanti.

È possibile assegnare un array ad un parametro di default.

Parametri Battezzati

Da PHP 8.0 è possibile passare un parametro in base al nome invece che che alla posizione.

<funzione>(<nome_param> : <valore>, …);

L’ordine in cui i paramatri sono passati non ha importanza. Il numero si.

L’uso dei parametri battezzati rende il codice più leggibile.

Tipizzazione dei Parametri

È possibile assegnare una tipologia di dato ai parametri di una funzione.

```
function <nome>( <tipo> $<parametro01>, …) {
    …;
}
```

L’interprete eseguirà un controllo ogni volta che viene chiamata la funzione.

Tipizzazione del Valore di Ritorno

È possibile assegnare una tipologia di dato al valore reso da una funzione.

```
function <nome>( <tipo> $<param01>, … ) :
   <type>  {
   …;
}
```

L’interprete eseguirà un controllo sul valore di ritorno all’uscita dalla funzione.

Rendere Valori Multipli

Una funzione rende sempre uno ed un solo valore. È possibile aggirare la limitazione facendo rendere un array creato al volo.

```
    ...;
    return [ 0, 1, 2 ];
}
```

La chiamata deve includere un'assegnazione:

```
[ $var01, $var02, $var03 ] = <funzione>( $<param01>, … );
```

Rendere un Reference

Perché una funzione renda un reference è necessario specificarlo sia nella dichiarazione che nell'assegnazione.

```
function &<funzione>( <tipo> $<param01>, …) {
    …;
}

$var01 =& <funzione>( <value01>, … );
```

Classi

Una classe definisce una tipologia di dato complesso.

Ogni classe può contenere le proprie costanti, variabili (Membri) e funzioni (Metodi) specifiche.

Una classe è definita mediante l'istruzione **class**.

```
class <nome> {
    ...;
}
```

Istanza di una Classe (Oggetto)

Per creare un'istanza di una classe si usa l'istruzione **new**.

```
$<istanza> = new <classe>();
```

Una classe deve essere definita prima di utilizzare gli oggetti / istanze.

Non esiste un limite agli oggetti di classe che si possono creare.

Istanza di una Classe (Oggetto)

All’interno del codice di una classe si può sempre accedere alle variabili ed ai metodi della stessa mediante la variabile predefinita **$this**.

Dall’esterno, le variabili ed i metodi di una classe possono essere acceduti mediante l’operatore **->** (Freccia).

Costruttore e Distruttore

Il costruttore di una classe viene chiamato per inizializzare ogni oggetto.

Il costruttore è una funzione membro avente nome standard __**construct**.

Il costruttore può accettare parametri.

Lo scopo del costruttore è allocare le risorse e portare l'oggetto in uno stato iniziale consistente.

Costruttore e Distruttore

Il distruttore di una classe viene chiamato quando viene distrutto l'ultimo reference ad un oggetto.

Il distruttore è una funzione membro avente nome standard **__destruct**.

Non sono previsti parametri per il distruttore.

Lo scopo del distruttore è liberare le risorse e pulire lo stato di un oggetto.

Visibilità di Membri e Metodi

Esistono 3 livelli di accesso alle variabili ed alle funzioni di una classe:

- **private**: Visibili solo dai membri di classe.
- **protected**: Visibili dai membri di classe e da quelli delle classi figlio.
- **public**: Visibili anche dall'esterno.

```
class logger  {
   public function log( $msg )  {
      echo $msg;
   }
}
```

Visibilità di Membri e Metodi

Gli oggetti della medesima tipologia possono accedere alle rispettive funzioni membro anche se sono dichiarate **private** o **protected**.

Costanti di Classe

PHP prevede la possibilità di definire costanti intere ad una classe. Le costanti possono essere definite aggiungendo l'istruzione **const** alla dichiarazione di una variabile.

```
const <COSTANTE> = <value>;
```

I valori delle costanti non possono essere modificati. L'interprete rende un errore se si prova a ridefinire una costante.

Ereditarietà

L’ereditarietà è una pratica assodata della programmazione ad oggetti che consente di ottenere classi derivate, che possono usare i metodi di quella d’origine, ridefinirli o aggiungerne di nuovi.

In PHP è possibile ereditare da una sola classe padre.

Per definire una classe derivata si usa l’istruzione **extends** nella dichiarazione.

Ereditarietà

```
class class_child extends class_parent  {
   // Redefine the parent method
   function var_display()  {
      echo "Extending class\n";
      parent::var_display();
   }
}
```

Parent

Una classe figlio può accedere ai metodi del padre usando l'istruzione **parent::**.

Il costruttore / distruttore di una classe figlio non chiama automaticamente quello della classe padre. La chiamata deve essere esplicita:

```
parent::__construct;

parent::__destruct;
```

Static

Dichiarare i membri o i metodi di una classe **static** li rende accessibili senza dover istanziare una classe.

```
class <nome_classe>  {
   public static $var_static = 'statica';
   ...;
   public static function metodo_static()  {
     ...;
   }
}
```

Static

È possibile accedere ad un membro oppure ad un metodo static di una classe utilizzando l'operatore **::** preceduto dal nome della classe.

```
<nome_classe>::metodo_static();

<nome_classe>::var_static;
```

Non possono invece essere acceduti tramite l'operatore **->**.

Self

La variabile **$this** non è disponibile. I metodi dichiarati static possono accedere ai soli membri static mediante l'operatore **self::**.

```
class <nome_classe>  {
   public static $var_static = 'statica';
   …;
   public static function metodo_static()  {
      return self::$var_static;
   }
}
```

Documentazione

Canali Web

- www.youtube.com/results?search_query=php
- www.youtube.com/playlist?list=PLP5MAKLy8lP_zqdyjNaPjh95NG40Op8he

Manuali

- www.php.net/manual/en/
- pear.php.net/manual/
- pecl.php.net/
- it.wikibooks.org/wiki/PHP
- learning.lpi.org/en/learning-materials/030-100/

Informazioni & Licenze