PHP 8

LA STORIA

PHP **nasce** nel 1994 con Rasmus Lerdorf **per tracciare il numero di visite all'interno della suo homepage personale**. Da qui il nome del PHP: Personal Home Page, che In seguito si trasformò in Hypertext Preprocessor.

AGGIORNAMENTI

Il progetto venne ripreso successivamente da Rasmus, riscritto e rilasciato integrando alcune funzionalità. A questo punto il **linguaggio** era **diventato** abbastanza **noto presso la community Open Source**, al punto che nel 1998 venne rilasciata la versione 3 che, alla fine dello stesso anno, coprirà circa il 10% dei server presenti in Rete. la versione corrente 5.x uscì nel luglio del 2004.

CARATTERISTICHE E VANTAGGI

LINGUAGGIO

Tipizzazione debole;
Non supporta i puntatori,
Utilizzato principalmente **per applicativi Web**,
Operazioni con le stringhe molto semplici da effettuare
Supporta I/O

Tipi di dati numerici supportati: 32-bit signed int, 64-bit signed int (long integer),
Tipi di dati float supportati: double precision float,
Non supporta array di dimensioni fisse,
Supporta array associativi,
consente di accedere in maniera semplicissima alle richieste HTTP di tipo GET e POST

GET E POST

GET è il metodo con cui vengono richieste la maggior parte delle informazioni ad un Web server, tali richieste vengono veicolate tramite query string, cioè la parte di un URL che contiene dei parametri da passare in input ad un'applicazione (ad esempio www.miosito.com/pagina-richiesta?id=123&page=3).

Il metodo POST, consente di inviare dati ad un server senza mostrarli in query string, è ad esempio il caso delle form.

CARATTERISTICHE E VANTAGGI

Importante è l'accesso in lettura/scrittura ai cookie del browser e il supporto alle sessioni sul server.

I cookie sono delle righe di testo usate per tenere traccia di informazioni relative ad un sito Web sul client dell'utente.

Inoltre anche se nato come linguaggio per Web, potrà essere utilizzato come linguaggio per lo scripting da riga di comando.
Uno dei punti di forza del PHP è la community molto attiva.
Sono migliaia le librerie di terze parti che ampliano le funzionalità di base e, nella maggior parte dei casi, sono tutte rilasciate con licenza Open Source.

Numerosissimi anche i framework sviluppati con questo linguaggio (Zend Framework, Symfony Framework, CakePHP, Laravel..) così come anche i CMS (WordPress, Drupal, Joomla, Magento, Prestashop..).
PHP è multipiattaforma, cioè può essere utilizzato sia in ambienti unix-based (Linux, Mac OSX) che su Windows.
La combinazione più utilizzata, però, resta quella LAMP ovvero Linux come sistema operativo, Apache come Web server, MySQL per i database e PHP

IDE (Integrated Development Environment)

Vengono utilizzati per l'autocompletamento del codice,
Gli IDE **sono software che aiutano i programmatori a sviluppare codice** attraverso l'autocompletamento delle funzioni, dei metodi delle classi, tramite la colorazione del codice o la segnalazione di errori di sintassi senza dover eseguire il codice per individuarli.
Si potrebbe preferire un IDE al semplice editor di testo.
Alcuni degli IDE più noti sono i seguenti, tutti multipiattaforma:

PHPStorm (premium);
Visual Studio Code
Eclipse (free);
Aptana Studio (free);
Zend Studio (premium).

PHPStorm è uno degli IDE più completi attualmente sul mercato.

I TAG E I COMMENTI

TAG DI APERTURA E DI CHIUSURA
Il tag di apertura del linguaggio <?php, mentre l'ultima il tag di chiusura ?>.
L'interprete PHP, infatti, **esegue solo il codice che è contenuto all'interno di questi due delimitatori.**
Questo consente di inserire del codice PHP all'interno di una normale pagina HTML così da renderla dinamica.

```
<?php
/*
In questo script vedremo quali sono gli elementi che compongono un file PHP. Questo ad esempio è un commento multilinea.
*/
// questo invece è un commento su singola riga
# questo è un altro tipo di commento
$nome = "TuoNome";
echo "Il mio nome è " .
$nome;
function stampa_nome($nome) {
    echo"<strong>Ciao " . $nome . "</strong>";
}
stampa_nome($nome);
?>
```

PSR-12

1. All PHP files MUST use the Unix LF (linefeed) line ending only. (quindi no CR+LF come DOS Style)
2. Ogni file PHP deve terminare SENZA UNA RIGA VUOTA ALLA FINE
3. The closing ?> tag MUST be omitted from files containing only PHP.

2.3 Lines

There MUST NOT be a hard limit on line length.

The soft limit on line length MUST be 120 characters.

Blank lines MAY be added to improve readability and to indicate related blocks of code except where explicitly forbidden.

There MUST NOT be more than one statement per line.

PSR-12

2.4 Indenting

Code MUST use an indent of 4 spaces for each indent level, and MUST NOT use tabs for indenting.

2.5 All PHP reserved keywords and types MUST be in lower case.

ex: break,callable,case,catch,class,....

Short form of type keywords MUST be used i.e. bool instead of boolean, int instead of integer etc.

3 La parte iniziale del file dovrebbe contenere alcuni blocchi tra cui la documentazione separati da una linea

```php
<?php

/**
 * This file contains an example of coding styles.
 */

declare(strict_types=1);

namespace Vendor\Package;

use Vendor\Package\{ClassA as A, ClassB, ClassC as C};
use Vendor\Package\SomeNamespace\ClassD as D;
use Vendor\Package\AnotherNamespace\ClassE as E;

use function Vendor\Package\{functionA, functionB, functionC};
use function Another\Vendor\functionD;

use const Vendor\Package\{CONSTANT_A, CONSTANT_B, CONSTANT_C};
use const Another\Vendor\CONSTANT_D;

/**
 * FooBar is an example class.
 */
class FooBar
{
    // ... additional PHP code ...
}
```

LE VARIABILI

Tipi Di Dato

Il PHP è un linguaggio con tipizzazione debole. Ciò significa che non vi sono regole molto rigide sulla dichiarazione del tipo di variabile. Ad esempio, linguaggi come il C, il C++, il C# o il Java prevedono che si dica sempre di che tipo sarà la variabile.

int a = 5;

Il PHP invece, non richiede alcun tipo e la variabile si dichiara in questo modo:

$a = 5;

L'unica accortezza richiesta è il simbolo del dollaro prima di ogni nome di variabile, utilizzato per segnalare che quell'elemento è una variabile.

E' possibile, inoltre, utilizzare numeri e lettere (ma non solo numeri) nel nome e il carattere underscore _ .

E' possibile quindi dichiarare variabili come:
$_var = 4;
$var1 = a;
$_1var = 33;

Non è possibile invece dichiarare variabili come:
$1234 = 33; $1var = a;
poiché iniziano con un numero.

Attenzione: il PHP è un linguaggio case sensitive (fa distinzione tra maiuscole e minuscole), perciò una variabile che inizia con la lettera maiuscola ed un'altra con la lettera minuscola sono considerate differenti.

PHP da linea di comando

PHP non è solo un linguaggio di sviluppo per il Web e **può essere utilizzato anche come soluzione per lo scripting da riga di comando.**

Prima di analizzare le modalità d'impiego e le sue potenzialità dobbiamo **assicurarci che l'eseguibile sia correttamente raggiungibile** dalla nostra shell.

Apriamo quindi **il Terminale e proviamo a lanciare il comando:**

php -v

In base alla versione di PHP che abbiamo installato sulla nostra macchina otterremo un risultato simile al seguente:

~ → php –v

```
PHP 8.2.4 (cli) (built: Mar 14 2023 17:54:25) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.4, Copyright (c) Zend Technologies
    with Xdebug v3.3.2, Copyright (c) 2002-2024, by Derick Rethans
```

Nel caso dovessimo ottenere un **errore** simile a command not found ciò significherebbe che **probabilmente la cartella dell'eseguibile non è stata inserita tra i path del sistema.**

PHP da linea di comando

Esecuzione di codice dalla command line
Usando il parametro -r possiamo far eseguire qualunque istruzione PHP al terminale:

```
> php -r 'echo "hello world";'
hello world
```

Esecuzione di script da file

Qualsiasi file PHP può essere eseguito direttamente da Terminale attraverso il comando:
php file.php
Al file è possibile passare anche dei parametri che possono essere utilizzati dallo script.

php index.php eee ddd

Per recuperare i parametri all'interno del nostro script PHP mette a disposizione 2 variabili globali:

1. $argv: un array che contiene gli argomenti;
2. $argc: un intero che indica il numero di argomenti contenuti nell'array $argv.

```php
<?php
/**
 * esempio di riga comando
 */
print 'argc:';
print_r($argc);

print PHP_EOL;
print 'argv:';
print PHP_EOL;
print_r($argv);

/*
argc:3
argv:
Array
(
    [0] => index.php
    [1] => eee
    [2] => ddd
)
*/
```

PHP E HTML

Il codice **contenuto all'interno dei tag** delimitatori di PHP viene **interpretato** e, di conseguenza, **viene stampato soltanto l'output** che noi abbiamo stabilito in sede di stesura del sorgente.

Sono le ore 15:04 del giorno 25/06/2022.

```
<!DOCTYPE html>
<html>
<head>
  <title><?php echo "Titolo della pagina"; ?></title>
</head>
<body>
  Sono le ore <?php echo date('H:i'); ?> del giorno <?php echo date('d/m/Y'); ?>.
</body>
</html>
```

FORMATTAZIONE DELLA PAGINA HTML

IL CARATTERE \n [importante utilizzare con doppio apice "\n"]

\n = PHP_EOL (entrambi producono new line)

Solitamente nei linguaggi di programmazione il carattere \n sta ad **indica**re **un ritorno a capo.**
Anche nel PHP il comportamento è lo stesso, ma bisogna **presta**re **attenzione anche al comportamento dell'HTML**. Il codice:
<?php echo "Ciao TuoNome,\n come stai?"; ?>
verrà infatti visualizzato su una sola riga nonostante il \n.
Visualizzando il sorgente della pagina, invece, il testo risulterà disposto su due righe.
**Nell'HTML il tag corretto per forzare un ritorno a capo è
 e non è sufficiente scrivere del testo su un'altra riga.**

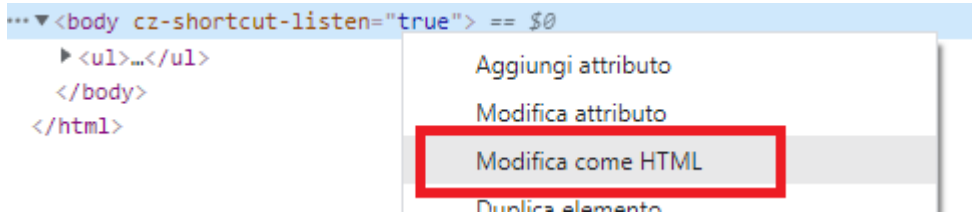
Per avere un funzionamento corretto (e cioè identico a quello atteso), abbiamo quindi bisogno di scrivere:<?php echo "Ciao Simone,
 come stai?"; ?>

FORMATTAZIONE DELLA PAGINA HTML

Se abbiamo bisogno di produrre un consistente quantitativo di markup HTML attraverso il codice PHP, può essere utile formattarlo in modo di aumentarne la leggibilità. Vediamo un esempio:
<?php echo "
\n
list item 1
\n
list item 2
\n
\n"; ?>

Il codice appena visto stamperà tramite il Web browser un codice HTML come il seguente:

list item 1
list item 2



senza \n sarebbe:

Tipi di Dato

PHP supporta I seguenti tipi:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL

Boolean – valori booleani (vero, falso)

Rappresentare i valori “vero” o “falso” all'interno di espressioni logiche.

```
<?php
   $vero = true;
   $falso = false;
   $vero = 1 && 1;
   $falso = 1 && 0;
   $vero = 1 || 0;
   $falso = 0 || 0;
?>
```

Integer – valori interi

I valori interi sono rappresentati da cifre positive e negative comprese in un intervallo che dipende dall'architettura della piattaforma (32 o 64 bit). I valori minimo e massimo sono rappresentati attraverso la costante PHP_INT_MAX.

32 bit:

Con segno: da −2.147.483.648 a +2.147.483.647

Senza segno: da 0 a +4.294.967.295

64 bit:

Con segno: da −9.223.372.036.854.775.808 a +9.223.372.036.854.775.807
Senza segno: da 0 a +18.446.744.073.709.551.615

```
<?php
   $intero = 1;
   $intero = 1231231;
   $intero = -234224;
   $intero = 1 + 1;
   $intero = 1 - 1;
   $intero = 3 * 4;
?>
```

```
Creare un file test.php
<?php
print PHP_INT_MIN . ", " .
PHP_INT_MAX;

.. e eseguirlo ..
>php test.php
```

```
-9223372036854775808, 9223372036854775807
```

Float o double – numeri in virgola mobile

Il tipo float in PHP, conosciuto anche come double, è un numero a virgola mobile positivo o negativo. Può essere specificato tramite il punto . o, in alternativa, un esponente.

I valori con cifre decimali sono rappresentati mediante l'utilizzo del **. e non** della **,** come nel nostro sistema.

```
<?php
   $float = 10.3;
   $float = -33.45;
   $float = 6.1e6; // 6,1 * 10^6 => 6.100.000
   $float = 3E-7;  // 3 * 10^-7  =>
3/10.000.000 = 0,0000003
?>
```

Bisogna prestare attenzione a **non usare questo tipo di dato per operazioni con valori precisi**, come nel caso in cui si ha a che fare con valute.

In quel caso bisognerebbe utilizzare invece le funzioni di **BCMath**.

```php
$float1 = 123.30; // Sintassi con il punto
$float2 = 120e-3; // Sintassi con esponente

echo "$float1\n"; // 123.30
echo "$float2\n"; // 0.12
```

float e BCMath

Floating point numbers have limited precision. Although it depends on the system, PHP typically uses the IEEE 754 double precision format, which will give a maximum relative error due to rounding in the order of 1.11e-16.

Quindi il float perde precision dopo una operazione e per I confronti meglio utilizzare: `bccomp`

bccomp Ritorna 0 se uguali, 1 se maggiore il primo, -1 se maggiore il secondo

Vedi esempio a destra

```
$a = 0.17;
$b = 1 - 0.83; //0.17
if ($a != $b) {
    print "A no match a:" . $a . "
b:" . $b;
 }else{
     print "A match";
 }
//stampa A no match a:0.17 b:0.17
if (bccomp($a, $b, 2) != 0) {
    print "B no match a:" . $a . "
b:" . $b;
}else{
    print "B match";
}
//stampa B match
```

Utilizzando bcadd

Vedi esempio a destra

```
$a = 0.17;
$b = bcadd(1, -0.83, 2); //0.17
if ($a != $b) {
    print "A no match a:" . $a . "
b:" . $b;
 }else{
     print "A match";
 }
 print "\n";
if (bccomp($a, $b, 2)) {
    print "B no match a:" . $a . "
b:" . $b;
}else{
    print "B match";
}
//A match
//B match
```

String – testi o stringhe di caratteri

Una stringa è un insieme di caratteri.

In PHP non è necessario definire la dimensione massima della stringa ma in ogni momento si può modificare il contenuto senza prima verificare se la variabile può contenere una determinata stringa.
Una stringa può essere **delimitata dal carattere ' (apice singolo) o dal carattere " (doppio apice).**
Il delimitatore utilizzato per determinare l'inizio di una stringa deve essere utilizzato anche per indicarne il suo termine.

```
<?php
   $stringa = "ciao come stai?";
   $stringa = 'ciao come stai?';
   $stringa = "lei mi ha chiesto 'come stai?'";
   $stringa = 'lei mi ha chiesto "come stai?"';
   //stringhe non valide perché contengono lo stesso carattere di apertura
   //all'interno della stringa
   $stringa_non_valida = "lei mi ha chiesto "come stai?"";
   $stringa_non_valida = 'lei mi ha chiesto 'come stai?';
   //utilizzare il backslash, \, per impiegare il carattere di apertura all'interno della stringa
   //tale operazione viene definita escaping delle stringhe
   $stringa_valida = "lei mi ha chiesto \"come stai?\"";
   $stringa_valida = 'lei mi ha chiesto \'come stai?\';
?>
```

Stampare stringhe con echo e print

Il costrutto **echo consente di stampare a schermo una o più stringhe**.

Essendo appunto **un costrutto**, e non una funzione, **non necessità di parentesi per essere richiamato.**

```
echo 'Questa è una stringa';
echo "Questa è una stringa";
echo "Questa è una stringa " . "concatenata";
echo "Questa è un'altra stringa ", "concatenata";
```

```
$nome = "TuoNome";
echo "Il mio nome è " . $nome;
echo "Il mio nome è ", $nome;
$array = array("TuoNome", "Luca");
echo "Il primo elemento dell'array si chiama ",
$array[0];
```

Stampare stringhe con echo e print

Possiamo anche inserire le variabili direttamente all'interno della stringa, senza concatenarle.

```
$nome = "TuoNome";
echo "Il mio nome è $nome";
```

In questo caso è necessario delimitare la stringa con il doppio apice " altrimenti non verrà interpretata la variabile.

Per accedere ad un elemento dell'array all'interno della stringa, oltre al doppio apice, abbiamo bisogno di delimitare l'accesso all'array con le parentesi graffe {}

```
$array = array("TuoNome", "Luca");
echo "Il primo elemento dell'array si chiama {$array[0]}";
```

echo in HTML e sintassi abbreviata

Un altro modo per utilizzare il costrutto echo è quello di utilizzare la sua sintassi abbreviata.

è necessario **aggiungere un = subito dopo l'apertura del tag PHP per stampare automaticamente il valore della variabile**.
Questa specifica alternativa, per funzionare correttamente, necessita che la direttiva short_open_tag sia abilitata all'interno del nostro php.ini in caso di versioni inferiori alla 5.4.0 del PHP.

Dalle versioni successive alla 5.4 , invece, <?= è sempre disponibile.

```
<?php
   //definizione variabili
   $nome = "TuoNome";
?>
... codice html ...
<p>Benvenuto <?=$nome?></p>
```

Stampare stringhe con print

Print è un altro costrutto di PHP utilizzato **per stampare** stringhe sullo standard output.

È leggermente meno performante del costrutto echo, ma il funzionamento è molto simile.
La differenza sostanziale tra i due costrutti è che quest'ultimo **ha un valore di ritorno booleano mentre il costrutto echo non ha valore di ritorno**.

Un'altra differenza non meno importante è che **print non permette di separare con la virgola diverse stringhe da stampare, ma consente unicamente di concatenarle tra di loro.**

OK:
```
echo "Questa è un'altra stringa ", "concatenata";
```

ERR:
```
print "Questa è un'altra stringa ", "concatenata";
```

Vari Esempi:

```
print "Questa è una stringa";
print "Questa è una stringa " . "concatenata";
// il seguente esempio produrrà un errore
print "Questa è un'altra stringa ", "concatenata";
$nome = "TuoNome";
print "Il mio nome è $nome";
$array = array(" TuoNome", "Luca");
print "Il primo elemento dell'array si chiama {$array[0]}";
```

Backslash

Il backslash \ deve essere utilizzato come carattere di escape quando si vuole includere nella stringa lo stesso tipo di carattere che la delimita;

se mettiamo un backslash davanti ad un apice doppio in una stringa delimitata da apici singoli (o viceversa), anche il backslash entrerà a far parte della stringa stessa.

Il backslash viene usato anche come “escape di sé stesso” nei casi in cui si desideri includerlo esplicitamente in una stringa.

```
<?php
   echo "Questo: \"\\\" è un backslash"; //
stampa: Questo: "\" è un backslash
   echo 'Questo: \'\\\' è un backslash'; //
stampa: Questo: '\' è un backslash
   echo "Questo: '\' è un backslash";    //
stampa: Questo: '\' è un backslash
   echo "Questo: '\\' è un backslash";   //
stampa: Questo: '\' è un backslash
?>
```

PER DELIMITARE UNA STRINGA

Per delimitare una stringa, sarà possibile **inserire delle variabili all'interno della stessa**.
In questo caso la variabile verrà automaticamente interpretate dal PHP e convertita nel valore assegnato ad essa (“esplosione della variabile”):
Utilizzare il " doppio apice

```php
<?php
    $nome = "TuoNome";
    $stringa = "ciao $nome, come stai?";
//ciao TuoNome, come stai?
    $stringa = 'ciao $nome, come stai?';
//ciao $nome, come stai?
//NON Funziona
?>
```

NOTAZIONE HEREDOC

La notazione heredoc, **utilizzata per specificare stringhe molto lunghe**, consente di **delimitare una stringa con i caratteri seguiti da un identificatore**;
a questo scopo in genere si usa **EOD**, ma è solo una convenzione, è possibile utilizzare qualsiasi stringa composta di caratteri alfanumerici e underscore, di cui il primo carattere deve essere non numerico (la stessa regola dei nomi di variabile).

Tutto ciò che segue questo delimitatore viene considerato parte della stringa, fino a quando non viene ripetuto l'identificatore seguito da un punto e virgola.

Attenzione: **l'identificatore di chiusura deve occupare una riga a sé** stante, deve iniziare a colonna 1 e non deve contenere nessun altro carattere (nemmeno spazi vuoti) dopo il punto e virgola.

La sintassi heredoc risolve i nomi di variabile così come le virgolette. Rispetto a queste ultime, con tale sintassi abbiamo il vantaggio di poter includere delle virgolette nella stringa senza farne l'escape:

```
<?php
$nome = "TuoNome";
$stringa = <<<EOD
Il mio nome è $nome
EOD;
echo $stringa;

$contenuto = <<<DATA;
Tutto questo contenuto
<div>verrà inserito nella
variabile </div>
DATA;

DATA; dalla 7.3 può non
essere ad inizio riga
```

ARRAY

Un array (o vettore) può essere considerato **come una matrice matematica.**

In PHP esso rappresenta una variabile complessa che restituisce una mappa ordinata basata sull'associazione di coppie chiave => valore.

La chiave di un array può essere un numero o di tipo stringa, mentre il valore può contenere ogni tipo di dato, compreso un nuovo array.

```
<?php
   $array = array(); //array vuoto
   $array = [];      //array vuoto nella nuova
sintassi
   $array = array('a', 'b', 'c');
   $array = array(1, 2, 3);
?>
```

Array

Possiamo descrivere un array come una matrice matematica.
In termini più semplici può essere considerato **una collezione di elementi ognuno identificato da un indice**; **gli indici** di un array **possono essere numerici o stringhe**.
Gli elementi dell'array possono contenere al proprio interno qualsiasi tipo di dato, compresi oggetti o altri array.
Un array può essere definito in due modi equivalenti:

```
$array = array();
$array = [];
```

Esempio:

```
$frutti = array('banana', 'pesca', 'lampone');
$frutti = ['banana', 'pesca', 'lampone'];
```

Array e la funzione print_r()

In ogni momento **possiamo visualizzare il contenuto di un array con la funzione print_r(),** utile in caso di debug.
Un array è quindi un contenitore di elementi contraddistinti da un indice.

Se l'indice non viene esplicitato di default è di tipo numerico 0-based.

0-based sta ad indicare che l'indice dell'array inizierà da 0 anziché da 1.

Come abbiamo potuto vedere nell'output dell'esempio precedente, il primo elemento dell'array (banana) possedeva infatti l'indice con valore 0.

```
$frutti = ['banana', 'pesca', 'lampone'];
print_r($frutti);
```

L'esempio restituirà un output come il seguente:

```
Array
(
[0] => banana
[1] => pesca
[2] => lampone
)
```

Array ad indici

sono array con indice numerico
è possibile accedere ai singoli elementi attraverso il proprio indice.

L'esempio stamperà Il mio frutto preferito è il lampone perché ho deciso di stampare il terzo (indice 2 partendo da 0) elemento dell'array.

```
$frutti = ['banana', 'pesca', 'lampone'];
echo "Il mio frutto preferito è il " . $frutti[2];
```

Array associativi

Un altro modo molto di utilizzare gli array è quello di **sfruttare gli indici come stringhe** anziché come valore numerico.
Nell'array appena definito **ogni elemento è caratterizzato da una coppia chiave -> valore** in cui la chiave (o indice) è il nome di una persona e il valore è il suo anno di nascita.

```
$annoDiNascita = [
'TuoNome' => '1986',
'Gabriele' => '1991',
'Giuseppe' => '1992',
'Renato' => '1988'
];
```

Per stampare la data di nascita di uno degli utenti:

```
echo "L'anno di nascita di Gabriele è " . $annoDiNascita['Gabriele'];
```

Usando la funzione print_r() sull'array definito in precedenza, eseguendo quindi il comando print_r($annoDiNascita); otterremo il seguente output:

```
Array
(
[TuoNome] => 1986
[Gabriele] => 1991
[Giuseppe] => 1992
[Renato] => 1988
)
```

Array multidimensionali

Un array può contenere ulteriori array e si definisce multidimensionale

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

```
$partecipanti = [
'TuoNome' => [
'anno' => '1986',
'sesso' => 'M',
'email' => 'test@notreal.com'
],
'Gabriele' => [
'anno' => '1991',
'sesso' => 'M',
'email' => 'test2@notreal.com'
],
'Josephine' => [
'anno' => '1985',
'sesso' => 'F',
'email' => 'test3@notreal.com'
],
];
```

Array multidimensionali

In questo caso **per accedere ad un valore specifico dell'array** possiamo usare la sintassi:

```
echo 'La mail di Josephine è ' .
$partecipanti['Josephine']['email'];
```

Per **Aggiungere o modificare il valore di un elemento**

```
$partecipanti[' TuoNome']['email'] = 'valid@email.com'
```

Nel caso in cui volessimo **aggiungere un nuovo partecipante:**

```
$partecipanti['Giuseppe'] = [
'anno' => 1992,
'sesso' => 'M',
'email' => 'giuseppe@test.com'
];
```

LE FUNZIONI

LE FUNZIONI

Una funzione viene dichiarata con la keyword function, seguita dal nome della funzione e, tra parentesi tonde, dal nome dei parametri necessari alla funzione. Il codice della funzione è delimitato da due parentesi graffe { ... }

```
<?php
/*In questo script vedremo quali sono gli elementi che compongono un file PHP. Questo ad esempio è un commento multilinea.
*/
// questo invece è un commento su singola riga
# questo è un altro tipo di commento
$nome = "TuoNome";
echo "Il mio nome è " . $nome;
function stampa_nome($nome) {
    echo "<strong>Ciao " . $nome ."</strong>";
}
stampa_nome($nome);
?>
```

PSR-12

A function declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

```
function fooBarBaz($arg1, &$arg2, $arg3 = [])
// NO BLANK LINE
{
// NO BLANK LINE
    // function body
// NO BLANK LINE
}
```

Quindi:

```
function fooBarBaz($arg1, &$arg2, $arg3 = [])
{
    // function body
}
```

LE FUNZIONI

Nel caso in cui, invece, **dovesse ritornare qualcosa, si utilizza la keyword** return.

Ad esempio, supponiamo di voler scrivere **una funzione che sommi due parametri** e restituisca il risultato, il codice PHP necessario sarà:

```
function somma($a, $b) {
  $somma = $a + $b;
  return $somma;
}
$somma = somma(3,5); // $somma sarà uguale ad 8
```

Funzioni in PHP

I parametri passati in ingresso ad una funzione possono anche essere opzionali se definiamo un valore di default.
Supponiamo che il secondo parametro della funziona sia opzionale e, se non definito, sia di default 10, potremmo a quel punto richiamare la nostra funzione passando solo il primo parametro.

Automaticamente PHP sommerà al nostro primo parametro il valore 10

```
function somma($a, $b = 10) {
    return $a + $b;
}
```

Il codice della funzione è lo stesso, l'unica cosa che cambia è il valore di inizializzazione della variabile $b.
Eseguendo il codice:
echo "La somma di 5 e 10 è: " . somma(5);
L'output sarà:
La somma di 5 e 10 è: 15

La keyword global

Se vogliamo **accedere al valore di una variabile globale all'interno di una funzione possiamo utilizzare la keyword global**.

Attraverso di essa, infatti, possiamo bypassare la limitazione dello scope globale.
Riprendendo l'esempio appena visto, richiamando la variabile attraverso global il nostro codice non genererà un errore.

```
$x = "scope globale";
function test_scope() {
    global $x;
    echo $x;   //stamperà scope globale
}
echo $x;   //stamperà scope globale
```

Scope delle variabili

Lo scope delle variabili può essere **di 3 tipi: locale-globale-statico**

LOCALE

Una **variabile dichiarata all'interno di una funzione** ha uno scope locale e non ha visibilità al di fuori di essa.

```
function test_scope() {
    $x = "scope locale";
    echo $x;  //stamperà scope
locale
}
echo $x; //genererà un errore
perché $x non è accessibile al di
fuori della funzione
```

GLOBALE

Una **variabile dichiarata al di fuori di una funzione** ha uno scope globale e non può essere accessibile all'interno di una funzione

```
$x = "scope globale";
function test_scope() {
    echo $x;   //genererà un errore perché $x
non è accessibile all'interno della funzione
}
echo $x;   //stamperà scope globale
```

Lo Scope statico

Quando **una funzione termina la sua esecuzione**, **il contenuto occupato dalle variabile dichiarate al suo interno viene liberato in memoria**.
Nel caso in cui non volessimo cancellare il suo contenuto possiamo dichiararla come static.

Vediamo due esempi, uno classico e uno che definisce la variabile come statica.

```
function test_static() {
    $x = 0;
    $x = $x+1;
    echo $x;
}
test_static(); //stamperà 1
test_static(); //stamperà 1
test_static(); //stamperà 1
```

```
function test_static() {
    static $x = 0;
    $x = $x+1;
    echo $x;
}
test_static(); //stamperà 1
test_static(); //stamperà 2
test_static(); //stamperà 3
```

Variabili superglobali

PHP fornisce alcune **variabili superglobali** che sono sempre disponibili in tutti gli scope

es:

```
<?php
echo($_SERVER['HTTP_HOST'])."<br>";
echo($_SERVER['HTTP_USER_AGENT'])."<br>";
echo($_SERVER['REMOTE_ADDR'])."<br>";
echo($_SERVER['SERVER_PROTOCOL'])."<br>";
echo($_SERVER['REQUEST_METHOD'])."<br>";
echo($_SERVER['QUERY_STRING'])."<br>";
```

```
print $GLOBALS['miavar'];
```

Variabile	Descrizione
`$GLOBALS`	Contiene le variabili definite come globali attraverso la keyword `global` .
`$_SERVER`	Contiene gli header e le informazioni relative al server e allo script.
`$_GET`	Contiene i parametri passati tramite URL (es http://sito.com/?param1=ciao&param2=test).
`$_POST`	Contiene i parametri passati come POST allo script (es.: dopo il submit di una form).
`$_FILES`	Contiene le informazioni relative ai file uploadati dallo script corrente attraverso il metodo POST.
`$_COOKIE`	Contiene i cookie.
`$_SESSION`	Contiene le informazioni relative alla sessione corrente.
`$_REQUEST`	Contiene tutti i parametri contenuti anche in `$_GET` , `$_POST` e `$_COOKIE` .
`$_ENV`	Contiene tutti i parametri passati all'ambiente.

Le Costanti in PHP define('NOME', "valore")

una porzione di memoria destinata a **contenere un dato** caratterizzato dal fatto di essere **immutabile durante l'esecuzione di uno script**.

Sulla base di quanto appena detto le costanti possono essere quindi considerate come **l'esatto opposto di una variabile**, inoltre, sempre a differenza delle variabili**, le costanti divengono automaticamente globali per l'intero script nel quale vengono definite**.
Una costante **può essere definita in PHP attraverso l'impiego del costrutto define che accetta due parametri in ingresso**, questi ultimi sono: **il nome della costante e il valore associato ad essa** in fase di digitazione del codice.

Le costanti in PHP vengono **dichiarate attraverso dei nomi che iniziano con una lettera o con l'underscore**, per esse **non è** quindi **previsto l'impiego del carattere iniziale $** come avviene invece per le variabili.

PSR-1
Per convenzione i nomi delle costanti vengono scritti interamente in maiuscolo.
Il valore contenuto da una costante può essere **soltanto di tipo scalare o null**.
I **tipi di dato scalare sono**: interi, float, stringhe e booleani

```
define('COSTANTE', 'stringa');
define('POST_PER_PAGINA', 10);
define('TEMPLATE_EMAIL', 'template.html');
```

```
define('NOME',123);
print NOME;
//stampa 123
```

Costanti: leggere il valore

Per accedere al valore contenuto dalla costante è sufficiente **utilizzare il suo nome all'interno del codice.**

```
echo COSTANTE;  //stamperà "stringa" in output
echo "Numero di posta per pagina: " . POST_PER_PAGINA;
```

COSTANTI defined('COSTANTE')

Per verificare che una costante sia stata effettivamente definita, si può **utilizza**re **la funzione** nativa di PHP chiamata **defined()**

```
echo defined('COSTANTE');  //stamperà "1" in quanto la costante "COSTANTE" esiste
echo defined('COSTANTE_NON_ESISTENTE'); //non stamperà nulla

var_dump(get_defined_constants(true));
```

COSTANTI get_defined_constants()

Per recuperare tutte le costanti utilizzate all'interno dell'esecuzione di uno script si ha la possibilità di **utilizza**re invece **la funzione** denominata tramite una sintassi get_defined_constants()

```
print_r(get_defined_constants(true));
```

Le Costanti Predefinite

Il linguaggio PHP, oltre a consentire la definizione di nuove costanti durante la digitazione dei sorgenti, contiene anche alcune **costanti** predefinite che potrebbero rivelarsi **utili in circostanze specifiche**:

Costante	Descrizione
`__FILE__`	Contiene il percorso (*path*) del file su cui ci troviamo.
`__DIR__`	Contiene il percorso della directory in cui è contenuto il file corrente.
`__FUNCTION__`	Contiene il nome della funzione che stiamo utilizzando.
`__LINE__`	Contiene il numero di riga corrente.
`__CLASS__`	Contiene il nome della classe corrente.
`__METHOD__`	Contiene il nome del metodo corrente.
`__NAMESPACE__`	Contiene il nome del namespace corrente.
`__TRAIT__`	Contiene il nome del trait corrente.

Le espressioni in PHP

In informatica **un'espressione** può essere considerata come **la combinazione di uno o più valori con operatori e funzioni**. Tale combinazione produce un valore come risultato.
Si può applicare la definizione matematica di espressione anche nei linguaggi di programmazione.
In PHP può essere definita espressione un qualsiasi costrutto che restituisce un valore

```
5 //il valore dell'espressione è 5
"stringa" // il valore dell'espressione è "stringa"
5 > 1; // il valore dell'espressione è true
5 < 1; // il valore dell'espressione è false
5 * 3 // il valore dell'espressione è 15
"ciao " . "TuoNome" // il valore dell'espressione è "ciao TuoNome"

il . concatena le stringhe

/*
pow() restituisce il primo parametro elevato alla potenza del secondo
il valore dell'espressione sarà: 2 + 4 = 6
*/
2 + pow(2, 2)
```

Le espressioni in PHP

Il valore di una qualsiasi espressione può essere assegnato ad una variabile.

Nel caso proposto di **seguito il valore della variabile è dato dal risultato dell'espressione che si trova a destra dell'operatore di assegnazione (=)**

Questa sintassi consente di avere più espressioni all'interno di una singola istruzione.

```
$a = 3 + 3; //il valore di $a è il risultato dell'espressione: $a = 6
```

OPERATORI ARITMETICI DI PHP

Altri operatori messi a disposizione da PHP per effettuare operazioni aritmetiche.

Quelli più semplici:

```
$a = 1 + 1; // operatore di somma
$b = 1 - 1; // operatore di sottrazione
$c = 1 * 1; // operatore di moltiplicazione
$d = 2 / 1; // operatore di divisione
```

Operatore modulo

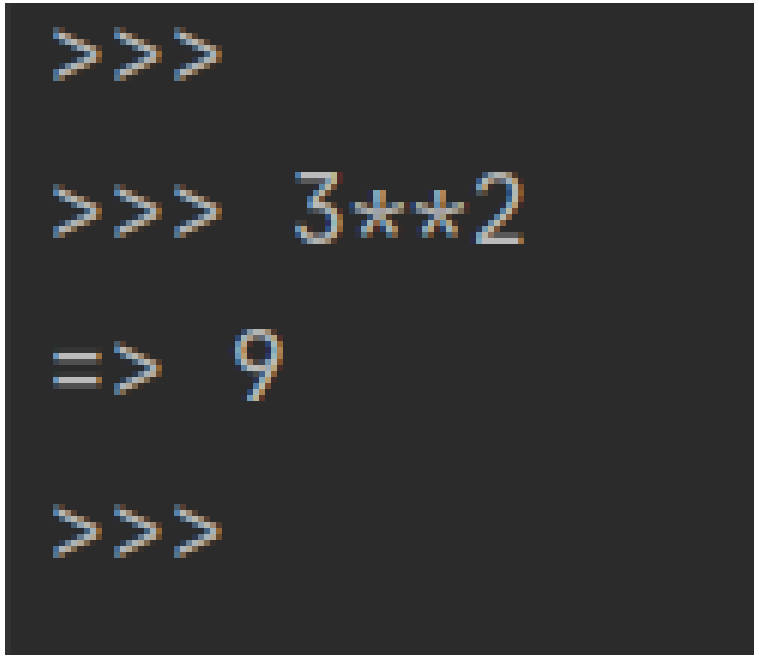
Abbiamo a disposizione anche **l'operatore modulo**, il quale **restituisce il resto di una divisione**
L'operatore modulo è **spesso utilizzato per determinare se un valore è pari o dispari.**
Nel caso riportato a destra la condizione stamperà a schermo 5 è dispari dato che il resto dell'operazione è 1 e non 0

```
$e = 5 % 2; // 5 diviso 2 = 2 con il resto di 1 => $e = 1
```

```
$e = 5;
if(($e % 2) == 0) {
    echo "5 è pari";
else {
    echo "5 è dispari";
}
```

Operatore Elevazione a Potenza **

**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power



Le espressioni in PHP

Ci sono **operatori che possono avere priorità maggiore e che quindi vengono eseguiti prima di altri.**

L'operatore di incremento ++ aumenta unitariamente il valore della variabile a cui è applicato.

Al termine della valutazione, quindi, verrà stampato a schermo il valore 2.

L'operatore può essere applicato prima (pre-incremento) o dopo (post-incremento) rispetto alla variabile di riferimento.

```
$i = 1;
echo ++$i;
//stampa 2;
```

A seconda della posizione cambia la priorità e, di conseguenza, il risultato dell'operazione.

Nel caso in cui l'operatore venga posizionato dopo la variabile, il suo incremento verrà posticipato al termine della valutazione dell'espressione.

In quest'ultimo esempio verrà stampato a schermo il valore 1

```
$i = 1;
echo $i++;
//stampa 1
```

Le parentesi

Le parentesi () modificano la priorità delle operazioni.

in un'espressione proprio come accade nella matematica, valutando prima il valore in esse contenuto e poi il resto dell'espressione.
Nel nostro caso viene prima valutato il valore del modulo e poi eseguito il resto dell'espressione.

```
$e = 5;
if(($e % 2) == 0) {
    echo "5 è pari";
else {
    echo "5 è dispari";
}
```

Operatori Aritmetici

Gli operatori aritmetici possono essere utilizzati non solo tra valori scalari ma anche con variabili e funzioni.
il risultato di un'espressione viene assegnato come risultato alla variabile stessa.
La variabile $b, quindi, viene modificata sommando il suo valore iniziale al valore di $a

```
$a = 1 + 1;
$b = 4 - 2;
$b = $b + $a; // 2 + 2 = 4
```

Operatori di assegnazione combinati

La stessa espressione appena vista può essere riscritta utilizzando gli operatori di assegnazione combinati:

Tali operatori possono essere utilizzati con le operazioni aritmetiche + - * / %

```
$b += $a; // equivale a $b = $b + $a
```

```
$b += $a; // $b = $b + $a
$b -= $a; // $b = $b - $a
$b *= $a; // $b = $b * $a
$b /= $a; // $b = $b / $a
$b %= $a; // $b = $b % $a
```

```
$str .= "aggiunto alla stringa";
```

Operatori di incremento e decremento

La differenza della posizione dell'operatore determina la priorità con cui viene eseguito l'incremento o il decremento.

Se si trova a sinistra viene eseguita prima di valutare il resto dell'espressione;
se si trova a destra viene eseguita dopo aver valutato l'espressione.

```
$i = 0;
++$i; // incrementa di 1 la variabile $i
$i++; // incrementa di 1 la variabile $i
--$i; // decrementa di 1 la variabile $i
$i--; // decrementa di 1 la variabile $i
```

Operatori logici a confronto

Gli operatori logici **consentono di determinare se la relazione** tra due o più espressioni **è vera** (TRUE) **o falsa** (FALSE).
L'unico operatore che **fa eccezione** a questa regola è **l'operatore not** che si occupa di negare il valore di un'espressione.

OPERATORE	NOME	ESEMPIO
and oppure &&	and	$x && $y
or oppure ||	or	$x || $y
xor	xor	$x xor $y
!	not	!$x

xor è vero se è vero a o è vero b, ma almeno uno vero e non entrambi

```
print (true xor true); // false
print "\n";
print (false xor true); // true
print "\n";
print (false xor false); // false
print "\n";
Mettere l'operazione tra ()
altrimenti non fa prima confronto
e poi print ma solo print e poi
confronto
```

Comportamento operatori logici

```
true && true;   //true
true && false;  //false
false && true;  //false
false && false;  //false
true || true;  //true
true || false;  //true
false || true;  //true
false || false;  //false
true xor true;  //false
true xor false;  //true
false xor true;  //true
false xor false;  //false
!true;  //false
!false;  //true
```

Espressioni e operatori logici

Nel caso in cui una delle espressioni sia intera, float o stringa e **nel caso i valori sono maggiori di "1", l'espressione viene interpretata come valore "1"**

```
1 && true;  //true
2 && true;  //true
"2" && true;  //true
0 && true;  //false
```

Nel caso in cui un'espressione sia un array, invece, essa viene valutata come vera se e soltanto se l'array non è vuoto, altrimenti è falsa:

```
array() && true;  //false
array(1, 2, 3) && true;  //true
```

Operatori di confronto

Gli operatori di confronto consentono di determinare il valore di due espressioni in base al confronto tra i loro valori.

Operator	Name
==	Equal
===	Identical
!=	Not equal
<>	Not equal
!==	Not identical
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<=>	Spaceship

operatore di confronto <=>

spaceship confronta due valori. Se il primo valore è maggiore del secondo restituisce un intero positivo 1; se il primo valore è minore del secondo restituisce un intero negativo -1; infine se i due valori sono uguali restituisce 0.

```
// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// Floats
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1
```

Operatori di uguaglianza == ATTENZIONE

```
0 == 'foobar' // true
in PHP 7

questo perché viene fatto il cast di 'foobar' come int e il
risultato sarà 0
0 == 0 è true

sarebbe stato meglio utilizzare
0 === 'foobar'
```

Confronto anche tra stringhe

I **confronti possono essere effettuati anche con** altri tipi di dato che non siano numerici, come ad esempio le **stringhe**.
Nel caso delle stringhe il confronto viene fatto in base all'ordine alfabetico dei caratteri seguendo l'ordine:

cifre.
caratteri maiuscoli;
caratteri minuscoli:

```
$a = 'MAIUSCOLO';
$b = 'minuscolo';
$c = '10 cifra';
$a > $b;  //falso perché la stringa inizia per un carattere maiuscolo
$b > $c;  //vero perché un carattere minuscolo è maggiore di una cifra
$c > $a;  //false perché un carattere maiuscolo è maggiore
'B' > 'A'; //vero perché la A viene prima della B
'm' > 'N'; //vero perché una lettera minuscola è maggiore di una MAIUSCOLA
```

If, else, istruzioni condizionali in PHP

Nel corso delle lezioni precedenti **è stato possibile analizzare alcuni frammenti di codice in cui venivano eseguite operazioni in base a determinate condizioni.**

Questo è possibile grazie alle istruzioni condizionali che ci consentono di avere comportamenti differenti all'interno del nostro codice in base a specifiche condizioni.

PSR-12

- There MUST be one space after the control structure keyword
- There MUST NOT be a space after the opening parenthesis
- There MUST NOT be a space before the closing parenthesis
- There MUST be one space between the closing parenthesis and the opening brace
- The structure body MUST be indented once
- The body MUST be on the next line after the opening brace
- The closing brace MUST be on the next line after the body
- elseif SHOULD be used instead of else if

```
if ( $expr1 ) {
    // if body
} elseif ($expr2) {

    NO!! neessuna linea vuota!!!!
    // elseif body
} else {
    // else body;

    NO!! neessuna linea vuota!!!!
}
```

if

Dopo la keyword if, deve essere indicata fra parentesi un'espressione da valutare (condizione). Questa espressione sarà valutata in senso booleano, cioè il suo valore sarà considerato vero o falso.
Dunque, **se la condizione è vera, l'istruzione successiva verrà eseguita**, altrimenti sarà ignorata.

Se l'istruzione da eseguire nel caso in cui la condizione sia vera **è una sola, le parentesi graffe sono opzionali**

```
if( <condizione> ) <istruzione>
```

```
if ($nome == 'Marco') {
  print "Marco";
}
```

```
if ($nome == 'Marco')
  print "Marco";
```

```
if ($i == 1) :
    print "si 1";
else :
    print "no 1";
endif;
```

Condizioni booleane

Se si valuta una stringa piena "ciao" **è considerato vero (TRUE).**

In sostanza, PHP considera come **falsi**:

- 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 esplicitamente.

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

```
if ("ciao") {
    print "è true";
}
```

If e Operatori di uguaglianza (==) e assegnamento (=)

$a = 7;

```php
<?php
if ($a = 4) echo "$a è uguale a 4!";
// 4 è uguale a 4!
```

Attenzione = è una assegnazione e non un confronto

viene stampato solamente perché $a è un numero != 0 e quindi vero

else

Per eseguire istruzioni se la condizione è falsa abbiamo bisogno del costrutto else
La parola chiave **else,** che **significa "altrimenti", deve essere posizionata subito dopo la parentesi graffa di chiusura** del codice previsto per il caso "vero" (o dopo l'unica istruzione prevista, se non abbiamo usato le graffe).

Anche per else valgono le stesse regole: niente punto e virgola, parentesi graffe obbligatorie se dobbiamo esprimere più di un'istruzione, altrimenti facoltative.

Ovviamente **il blocco di codice specificato per else verrà ignorato quando la condizione è vera, mentre verrà eseguito se la condizione è falsa**.

```
if (<condizione>) {
   <codice>
} else {
   <codice>
}
```

if – else - Istruzioni nidificate

Le istruzioni if possono essere nidificate una dentro l'altra, consentendoci così di creare codice di una certa complessità:

```
if ($nome == 'Luca') {
    if ($cognome == 'Rossi') {
        echo "Luca Rossi è di nuovo fra noi";
    } else {
        echo "Abbiamo un nuovo Luca!";
    }
} else {
    echo "ciao $nome!";
}
```

elseif

Un'ulteriore possibilità che ci fornisce l'istruzione if in PHP è quella di utilizzare la parola chiave **elseif**.
Attraverso quest'ultima possiamo indicare una seconda condizione, da valutare solo nel caso in cui quella precedente risulti falsa.
Indicheremo quindi, di seguito, il codice da eseguire nel caso in cui questa condizione sia vera, ed eventualmente, con else, il codice previsto per il caso in cui anche la seconda condizione sia falsa.

```
if ($nome == 'Luca') {
    echo "bentornato Luca!";
} elseif ($cognome == 'Verdi') {
    echo "Buongiorno, signor Verdi";
} else {
    echo "ciao $nome!";
}
```

Istruzione Switch

Lo switch che permette di **racchiude**re in un unico blocco di codice **diverse istruzioni condizionali.**

PSR-12

- The case statement MUST be indented once from switch
- The break keyword (or other terminating keywords) MUST be indented at the same level as the case body.
- There MUST be a comment such as // no break when fall-through is intentional in a non-empty case body.

```
switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}
```

istruzione con if, elseif ed else :

```
$colore = 'rosso';
if ($colore == 'blu') {
    echo "Il colore selezionato è blu";
} elseif ($colore == 'giallo') {
    echo "Il colore selezionato è giallo";
} elseif ($colore == 'verde') {
    echo "Il colore selezionato è verde";
} elseif ($colore == 'rosso') {
    echo "Il colore selezionato è rosso";
} elseif ($colore == 'arancione') {
    echo "Il colore selezionato è arancione";
} else {
    echo "Nessun colore corrispondente alla tua selezione";
}
```

Con Switch

```
$colore = 'rosso';
switch ($colore) {
    case 'blu':
        echo "Il colore selezionato è blu";
        break;
    case 'giallo':
        echo "Il colore selezionato è giallo";
        break;
    case 'verde':
        echo "Il colore selezionato è verde";
        break;
    case 'rosso':
        echo "Il colore selezionato è rosso";
        break;
    case 'arancione':
        echo "Il colore selezionato è arancione";
        break;
    default:
        echo "Nessun colore corrispondente alla tua selezione";
        break;
}
```

switch

il costrutto switch **contiene tra parentesi l'espressione da verificare e al suo interno** una serie di case che rappresentano **i possibili valori da valutare.**

Al termine di ogni case, inoltre, viene inserita la keyword break che **ci farà uscire** dall'esecuzione dell'istruzione dallo switch.

Tale keyword è importante perché, senza di essa, verrebbero valutati anche gli altri case contenuti nello switch, incluso **il default che è quello valutato quando nessuno degli altri case soddisfa la condizione.**

```
$colore = 'rosso';
switch ($colore) {
    case 'blu':
        echo "Il colore selezionato è blu";
        break;
    case 'giallo':
        echo "Il colore selezionato è giallo";
        break;
    case 'verde':
        echo "Il colore selezionato è verde";
        break;
    case 'rosso':
        echo "Il colore selezionato è rosso";
        break;
    case 'arancione':
        echo "Il colore selezionato è arancione";
        break;
    default:
        echo "Nessun colore corrispondente alla tua selezione";
        break;
}
```

switch e sequenze di case

I case possono anche essere sequenziali quindi, **nel caso in cui diversi valori contengono lo stesso codice da eseguire, potranno essere inseriti in sequenza**:

Nell'esempio presentato lo switch esegue la stessa istruzione in casi differenti.

Quindi, se il colore è “blu” o “verde”, allora visualizzeremo il messaggio relativo ai colori freddi, se il colore è “giallo”, “rosso” o “arancione” verrà visualizzato il messaggio relativo ai coloro caldi.

Se invece il colore non viene identificato da nessuno dei casi, verrà mostrato un messaggio di errore.

```
$colore = 'rosso';
switch ($colore) {
    case 'blu':
    case 'verde':
        echo "Il colore selezionato è un colore freddo";
        break;
    case 'giallo':
    case 'rosso':
    case 'arancione':
        echo "Il colore selezionato è un colore caldo";
        break;
    default:
        echo "Nessun colore corrispondente alla tua selezione";
        break;
}
```

Operatore ternario

L'operatore ternario rappresenta un'altra alternativa sintattica al costrutto if/else, esso è così chiamato perché è formato da **tre espressioni:**

```
($altezza >= 180) ? 'alto' : 'normale';
```

il valore restituito è quello della seconda o della terza di queste espressioni, a seconda che la prima sia vera o falsa.
In pratica, **si può considerare, in certi casi, una maniera molto sintetica di effettuare un costrutto condizionale basato su if.**
Questa espressione prenderà il valore “alto” se la variabile $altezza è maggiore o uguale a “180”, “normale” nel caso opposto.

L'espressione condizionale è contenuta fra parentesi e seguita da un punto interrogativo, **mentre due punti separano la seconda espressione dalla terza**.

Questo costrutto può essere utilizzato, ad esempio, per valorizzare velocemente una variabile senza ricorrere all'if.

PSR-12

The conditional operator, also known simply as the ternary operator, MUST be preceded and followed by at least one space around both the ? and : characters:

```
$variable = $foo ? 'foo' :
'bar';

$variable = $foo ?: 'bar';
```

When the middle operand of the conditional operator is omitted, the operator MUST follow the same style rules as other
binary comparison operators:

?: corrisponde al valore se falso

Operatore Colaesce ?? [ternary + isset]

null coalescing operator (??) è disponibile da PHP 7.
unisce le funzionalità di isset e dell'operatore ternario
Ritorna il primo operando se esiste ed è diverso da NULL, altrimenti ritorna il secondo operando.

```
$username = $_GET['username'] ?? 'not passed'; print($username); print("<br/>");
```

If else

```
if ($altezza >= 180)
$tipologia = 'alto';
else
$tipologia = 'normale';
```

Operatore ternario

```
$tipologia = ($altezza >= 180) ? 'alto' : 'normale';
```

utile nel caso in cui si necessiti rendere più compatto il codice

Operatore ternario abbreviato ?:

la variabile $messaggio conterrà il valore relativo alla “username” dell'utente soltanto nel caso in cui la variabile associata ad essa ($username) sia stata effettivamente definita, in caso contrario verrà utilizzata la stringa utente.

```
$messaggio = "Ciao " . ( $username ? $username : 'utente' );
```

Se il valore da restituire nel caso in cui l'espressione sia vera dovesse corrispondere esattamente all'espressione valutata, allora potrà essere omesso il secondo parametro unendo il punto interrogativo con i due punti, ?:, ottenendo così quello che viene chiamato l'operatore ternario abbreviato

```
$messaggio = "Ciao " . ( $username ?: 'utente' );
```

Operatori ternari concatenanti

è possibile concatenare operatori ternari tra di loro. Nell' esempio proposto abbiamo due operatori ternari:

il primo verifica che il valore associato alla variabile $età sia maggiore o uguale a “18”,

il secondo viene invece valutato soltanto se la prima condizione risulta vera, verificando che l'esame sia stato superato.

La variabile $patente **sarà quindi vera se e solo se si verificano entrambe le condizioni**, altrimenti sarà falsa.

```
$patente = ($eta >= 18) ? ($esame_superato ? true : false) : false;
```

Per quanto ci venga concessa la possibilità di concatenare gli operatori ternari, però, è sconsigliabile l'utilizzo in questa maniera dato che riduce di molto la leggibilità del codice.

I Cicli PHP, for, while e do

I cicli sono un altro degli elementi fondamentali di qualsiasi linguaggio di programmazione in quanto **permettono di eseguire determinate operazioni in maniera ripetitiva.**
Si tratta di una necessità che si presenta molto frequente.

FOR

Si ipotizzi di voler mostrare i multipli da 1 a 10 di un numero intero, ad esempio “5”.

La prima soluzione disponibile è basata sull'impiego del ciclo for.

Questo costrutto, simile a quello usato in altri linguaggi, utilizza la parola chiave **for, seguita,** fra parentesi**, dalle istruzioni per definire il ciclo**; **successivamente** si racchiudono **fra parentesi graffe tutte le istruzioni che devono essere eseguite ripetutamente**.

```
for ($mul = 1; $mul <= 10; ++$mul) {
$ris = 5 * $mul;
echo "5 * $mul = $ris <br/>";
}
```

For

Le tre istruzioni inserite *fra le parentesi tonde e separate da punto e virgola* **vengono trattate in questo modo:**

- **la prima viene eseguita una sola volta, all'inizio del ciclo;**
- **la terza viene eseguita alla fine di ogni iterazione del ciclo;**
- **la seconda deve essere una condizione, e viene valutata prima di ogni iterazione del ciclo;**

Quando la condizione risulta falsa, l'esecuzione del ciclo viene interrotta e il controllo passa alle istruzioni presenti dopo le parentesi graffe.

Ovviamente è possibile che tale condizione risulti falsa fin dal primo test: in questo caso, le istruzioni contenute fra le parentesi graffe non saranno eseguite nemmeno una volta.

Il formato standard è quindi quello che utilizza le parentesi tonde per definire un "contatore": con la prima istruzione lo si inizializza, con la seconda lo si confronta con un valore limite oltre il quale il ciclo deve terminare, con la terza lo si incrementa dopo ogni esecuzione.

```
for ($mul = 1; $mul <= 10; ++$mul) {
    $ris = 5 * $mul;
    echo "5 * $mul = $ris <br/>";
}
```

PSR-12

A for statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php

for ($i = 0; $i < 10; $i++) {
    // for body
}
```

Il ciclo PHP while

Il ciclo **while**, **rispetto al for, non mette a disposizione le istruzioni per inizializzare e per incrementare il contatore**, quindi dobbiamo **inserire queste istruzioni nel flusso generale del codice**, per cui mettiamo l'inizializzazione prima del ciclo e l'incremento all'interno del ciclo stesso, in fondo.

Anche in questa situazione il concetto fondamentale è che **l'esecuzione del ciclo termina quando la condizione fra parentesi non è più verificata**. Ancora una volta è possibile che il ciclo non venga mai eseguito nel caso in cui la condizione risulti falsa fin da subito.

```
$mul = 1;
while ($mul <= 10) {
  $ris = 5 * $mul;
  print("5 * $mul = $ris<br>");
  $mul++;
}
```

PSR-12

A while statement looks like the following. Note the placement of parentheses, spaces, and braces

```php
<?php

while ($expr) {
    // structure body
}
```

Il ciclo PHP do...while

Per assicurarci che il codice indicato tra le parentesi graffe venga eseguito almeno una volta: do...while

Con questa sintassi, il while viene spostato dopo il codice da ripetere, ad indicare che la valutazione della condizione viene eseguita solo dopo l'esecuzione del codice fra parentesi graffe.

Nel caso dell'esempio mostrato, abbiamo inizializzato la variabile $mul col valore “11” per creare una situazione nella quale, con i cicli visti prima, non avremmo ottenuto alcun output, mentre con l'uso del do...while il codice viene eseguito una volta nonostante la condizione indicata fra parentesi sia falsa fin dall'inizio. L'esempio stamperà quindi “5 * 11 = 55“.

```
$mul = 11;
do {
$ris = 5 * $mul;
print("5 * $mul = $ris<br>");
$mul++;
} while ($mul <= 10)
```

PSR-12

statement looks like the following. Note the placement of parentheses, spaces, and braces.

Expressions in parentheses MAY be split across multiple lines, where each subsequent line is indented at least once.

```
<?php

do{
    // structure body;
}while($expr);

<?php

do {
    // structure body;
} while (
    $expr1
    && $expr2
);
```

Ciclo foreach in PHP

Il foreach, permette di **cicla**re anche **gli array senza effettuare controlli sulla loro dimensione**.

PHP si occuperà automaticamente di terminare il ciclo quando tutti gli elementi sono stati processati.

Il costrutto dopo la parola chiave **foreach include** all'interno delle parentesi tonde tre elementi:

1. **l'array da ciclare**
2. **la parola chiave as**
3. **la variabile che contiene il valore dell'indice corrente**

Come i cicli finora descritti, anche il foreach conterrà il codice da eseguire ad ogni iterazione all'interno delle parentesi graffe.

un esempio che calcola la somma degli elementi di un array:

```
$array_monodimensionale = array(1, 2, 3, 4, 5);
$somma = 0;
foreach ($array_monodimensionale as $valore) {
$somma += $valore;
}
echo "La somma degli elementi dell'array è: " . $somma;
```

Foreach e array associativi

Abbiamo visto come sia facile ciclare gli elementi di un array ad una sola dimensione.

Ma un array può essere anche associativo.
Vediamo quindi come ciclare un array associativo:

Nell'esempio precedente stampiamo per ogni utente il nome e l'età di ognuno e, contestualmente, sommiamo le età di tutti.

Il foreach ha una sintassi identica a quella dell'esempio precedente con la differenza che**, in caso di array associativo, si utilizzano** due variabili: **la** variabile che identifica l'indice dell'array (nel nostro caso il nome dell'utente) **e la** variabile che identifica il valore dell'indice (nel nostro caso l'età).

```
$eta_utenti = array(
'TuoNome' => 29,
'Josephine' => 30,
'Giuseppe' => 23,
'Renato' => 26,
'Gabriele' => 24
);
$somma_eta = 0;
foreach ($eta_utenti as $nome => $eta) {
echo "L'utente " . $nome . " ha " . $eta . "
anni\n";
$somma_eta += $eta;
}
echo "La somma delle età degli utenti è: " .
$somma_eta;
```

Variabili passate per valore o per riferimento

Anteponendo il simbolo & davanti alla variabile passata ad una funzione o interna ad un ciclo, questa viene passata per riferimento e non per valore, questo significa che il php non lavora sulla **copia in memoria dell'array ma su**lla variabile **originale.**

```
$eta_utenti = array(
'TuoNome' => 29,
'Josephine' => 30,
'Giuseppe' => 23,
'Renato' => 26,
'Gabriele' => 24
);
foreach ($eta_utenti as $nome => &$eta) {
$eta++;
}
print_r($eta_utenti);
```

```
Array
(
    [TuoNome] => 30
    [Josephine] => 31
    [Giuseppe] => 24
    [Renato] => 27
    [Gabriele] => 25
)
```

PSR-12

A foreach statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php

foreach ($iterable as $key => $value) {
    // foreach body
}
```

Break

Il costrutto break **consente** di interrompere un ciclo e di uscirne senza continuare le iterazioni rimanenti.

È molto utile, ad esempio, quando si cerca un valore all'interno di un array: nel momento in cui l'elemento viene trovato non è necessario continuare con altri gli elementi.
nell'esempio appena proposto vogliamo visualizzare la posizione di arrivo dell'atleta Josephine. Quello che facciamo è utilizzare il costrutto foreach per ciclare su tutti gli atleti e verifichiamo per ognuno di essi il nome.

Non appena troviamo l'utente che stiamo cercando non è più necessario continuare le iterazioni per i restanti atleti ma possiamo interrompere l'esecuzione del ciclo. Per raggiungere il nostro scopo è sufficiente utilizzare la struttura di controllo break.

```
$atleti = array( 'TuoNome', 'Josephine', 'Giuseppe',
'Gabriele' );
$posizione_di_arrivo = 0;
foreach ($atleti as $posizione => $nome) {
    if ($nome == 'Josephine') {
        $posizione_di_arrivo = $posizione + 1;
        break;
    }
}
echo "Josephine è arrivata in posizione numero " .
$posizione_di_arrivo;
```

Continue

La struttura di controllo continue è utilizzata **per saltare il ciclo corrente e proseguire con le iterazioni successive** previste.
Nel nostro esempio, quindi, nel caso in cui un valore del ciclo non è divisibile per 2, saltiamo l'iterazione e passiamo direttamente a quella successiva.

```
for ($posizione=1; $posizione <= 10; $posizione++) {
    if ($posizione % 2 == 1) {
        continue;
    }
    echo "Il numero " . $posizione . " è un numero pari\n";
}
```

Gestire le stringhe

PHP mette a disposizione un set completo di funzioni che permettono di manipolare ed eseguire le operazioni sulle stringhe.

Analizziamo quelle utilizzate più frequentemente.
Per verificare la lunghezza di una stringa possiamo ricorrere alla funzione strlen() che restituirà il numero dei caratteri che la compongono spazi inclusi.

strlen() lunghezza di una stringa

```
strlen(string $string): int
```

```
$stringa = 'Stringa di esempio';
echo strlen($stringa); // restituirà 18
```

substr() estrazione di parte di stringa

```
substr(string $string, int $offset, ?int $length = null): string
```
Returns the portion of string specified by the offset and length parameters.

Immaginiamo, ad esempio, di **voler estrarre i vari elementi di una data,**
possiamo **utilizzare la funzione** substr() che prende in ingresso 3 parametri:

Parametro	Descrizione
$string	Una stringa.
$start	Carattere di iniziale, 0 di default. Consente di utilizzare un intero negativo per recuperare la porzione di stringa a partire dalla fine anziché dall'inizio.
$length	Numero di caratteri. Opzionale, se omesso viene restituita tutta la stringa a partire dal carattere iniziale. Consente di utilizzare un intero negativo, in quel caso ometterà `length` caratteri dalla fine della stringa.

Esempi

```
$data = '01/02/2016';
$giorno = substr($data, 0, 2); // 01
$mese   = substr($data, 3, 2); // 02
$anno   = substr($data, 6);    // 2016
$giorno = substr($data, -10, 2); // 01
$mese   = substr($data, -7, 2); // 01
$anno   = substr($data, -4, 4); // 01
```

substr_count ()

```
substr_count(
    string $haystack,
    string $needle,
    int $offset = 0,
    ?int $length = null
): int
```

La funzione substr_count() conta il numero di volte in cui una sottostringa si verifica in una stringa.

Nota: la sottostringa fa distinzione tra maiuscole e minuscole.

Nota: questa funzione non conta le sottostringhe sovrapposte.

```
<?php
echo substr_count("Hello world. The world
is nice","world");
// 2

?>
```

substr_replace()

sostituisce la parte di stringa oltre l'indice specificato

La funzione substr_replace() sostituisce una parte di una stringa con un'altra stringa.
Nota: se il parametro di inizio è un numero negativo e la lunghezza è minore o uguale a inizio, la lunghezza diventa 0.

Nota: questa funzione è a sicurezza binaria.

```
echo substr_replace("Hello worldddd
altro","earth",6);
//spampa Hello earth

//praticamente ha sostituito worldddd altro
```

```
<?php
echo substr_replace("Hello WORLD
","world",0); // 0 will start replacing at the
first character in the string
?>
```

Output:
world

strpos() ricerca di sottostringhe

la funzione strpos() restituisce la posizione della prima occorrenza della stringa ricercata oppure false nel caso non venga trovata alcuna occorrenza.

3 parametri in ingresso:

Parametro	Descrizione
$haystack	La stringa su cui effettuare la ricerca.
$needle	La porzione di stringa da ricercare.
$offset	Default a 0, indica la posizione da cui iniziare la ricerca.

Esempi

$stringa = 'La mia data di nascita è il 07 dicembre 1986';
echo strpos($stringa, '1986'); // **restituirà** 41 che è **la posizione in cui inizia la stringa** 1986
echo strpos($stringa, '1987'); // restituirà false perché non ci sono occorrenze

strpos() posizione di una stringa

La funzione **strpos** di PHP restituisce la posizione numerica della prima occorrenza di *cosa_cercare* (*needle*) all'interno della stringa *dove_cercare* (*haystack*). Se non viene trovata alcuna occorrenza strpos restituirà FALSE oppure ""

```
$stringa = 'Simone è nato nel 1986';
if (strpos($stringa, 'Simone') !== false) {
    echo 'Il tuo nome è Simone';
}
$strPos = (strpos($stringa, 'Simone') ) ?:
'non trovata'
print "la stringa è in posizione {$strPos}"
```

strrev() string reverse

INVERTIRE UNA STRINGA

Per effettuare l'inversione (reverse) di una stringa possiamo utilizzare la funzione strrev()

```
echo strrev("Hello world!"); // stamperà "!dlrow olleH"
```

str_replace() sostituire sottostringhe

SOSTITUIRE TUTTE LE OCCORRENZE DI UNA STRINGA

Nel caso in cui si voglia **sostituire una porzione di stringa all'interno di un'altra** stringa possiamo **utilizza**re la funzione str_replace(), essa prende in ingresso 4 parametri:

Parametro	Descrizione
$search	Il valore da cercare.
$replace	Il valore con cui sostituirlo.
$subject	La stringa o le stringhe in cui effettuare la sostituzione.
$count	Opzionale, variabile in cui memorizzare il numero di occorrenze sostituite.

```
print "str_replace:" .
str_replace("stringa", "testo", "mia
stringa da rimpiazzare");

//str_replace:mia testo da rimpiazzare
```

addslashes ()

La funzione **addslashes() restituisce una stringa con barre rovesciate davanti a:**

single quote (')
double quote (")
backslash (\)
NUL (the NUL byte)

```
<?php
$str = addslashes("Sto inserendo un ' apice in una stringa \ ");
echo($str);
?>

//Sto inserendo un \' apice in una stringa \\
```

chr()

La funzione **chr() restituisce un carattere dal valore ASCII specificato.**
Il valore ASCII può essere specificato in valori decimali, ottali o esadecimali.
I valori ottali sono definiti da uno 0 iniziale, mentre i valori esadecimali sono definiti da uno 0x iniziale.

```
<?php
echo chr(52) . "<br>"; // Decimal value
echo chr(052) . "<br>"; // Octal value
echo chr(0x52) . "<br>"; // Hex value
?>
```

https://www.ibm.com/docs/en/aix/7.2?topic=adapters-ascii-decimal-hexadecimal-octal-binary-conversion-table

Le richieste HTTP (GET E POST)

La specifica HTTP definisce 9 tipi di metodi alcuni dei quali non sono però usati o supportati da PHP; i più diffusi restano sicuramente GET e POST.

GET è il metodo con cui vengono richieste la maggior parte delle informazioni ad un Web server, tali richieste vengono veicolate tramite query string, cioè la parte di un URL che contiene dei parametri da passare in input ad un'applicazione.

Il metodo POST, invece, consente di inviare dati ad un server senza mostrarli in query string, è ad esempio il caso dei form

Le richieste HTTP (GET E POST)

Iniziamo con il metodo GET. Sicuramente è il più semplice e il più immediato.

È consigliato soprattutto in quelle richieste in cui è utile salvare nell'URL i parametri richiesti.

Per poter accedere ai parametri in GET di una richiesta HTTP proveniente da un form di ricerca avremo bisogno di due file:
form.html e
search.php.

Analizziamo innanzitutto il codice del file form.html che conterrà il form:

```
<!DOCTYPE html>
<html>
<head>
   <title></title>
</head>
<body>
   <form action="search.php">
     <input type="text" name="author" placeholder="Inserisci autore" />
     <input type="submit" value="Cerca" />
   </form>
</body>
</html>
```

Ricevere un parametro $_GET

```
$author = $_GET['author']; $author = filter_var($author,
FILTER_SANITIZE_STRING);
```

FILTER_SANITIZE_STRING DEPRECATO DALLA VERSIONE 8 USARE htmlspecialchars

```
$new = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
echo $new; // &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
```

POST

Il metodo POST si differenza da GET in quanto i parametri della richiesta non vengono passati in query string e quindi non possono essere tracciati nemmeno negli access log dei web server.

Caso d'uso comune di una richiesta in POST è un form che invia dati personali, come in una registrazione.
Vediamo quindi come accedere ai parametri POST con un esempio di registrazione tramite username e password.
Anche in questo caso abbiamo bisogno di due file: form.html e register.php.

Il codice è simile all'esempio precedente. Le differenze sostanziali sono la presenza di due campi username e password e, soprattutto, l'aggiunta dell'attributo method nel tag form.
Quando abbiamo bisogno di effettuare una richiesta POST è necessario specificare il metodo nel form.

Il file contenente il form:

```
<!DOCTYPE html>
<html>
<head>
   <title></title>
</head>
<body>
   <form action="register.php" method="post">
      <input type="text" name="username"
placeholder="Inserisci lo username" /><br>
      <input type="password" name="password"
placeholder="Inserisci la password" /><br>
      <input type="submit" value="Registrati" />
   </form>
</body>
</html>
```

Recevere un parametro $_POST

```
$username = $_POST['username'];
$password = $_POST['password'];
$username = filter_var($username, FILTER_SANITIZE_STRING); $password = filter_var($password, FILTER_SANITIZE_STRING); if (!$username || !$password) { $error = 'Username e password sono obbligatori'; }

if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }
```

post e validate

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```