PHP 7 - 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 ...
}
```

I COMMENTI

PHP supporta tre tipologie di commenti, un commento su più righe **(multilinea)** e consiste nel **racchiudere il testo** del commento **tra /* ... */**.
Tutto quello che è delimitato dai due tag non verrà interpretato dal PHP.
Le altre due modalità sono simili e consentono di inserire commenti su una singola riga attraverso l'utilizzo di **//**, doppio slash, **o** del carattere cancelletto, **#**.
Ovviamente il commento non deve trovarsi per forza all'inizio della riga ma anche dopo una qualsiasi istruzione PHP.

Ad esempio:echo "Ciao"; // stampo la parola Ciao

```
<?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);
?>
```

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 7.1.7 (cli) (built: Jul 15 2017 18:08:09) (NTS)
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2017 Zend Technologies

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.**

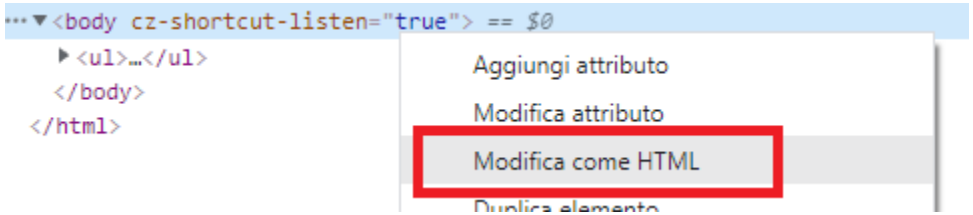
Per avere un funzionamento corretto (e cioè identico a quello atteso), abbiamo quindi bisogno di scrivere:<?php echo "Ciao Simone,
 come stai?"; ?>

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

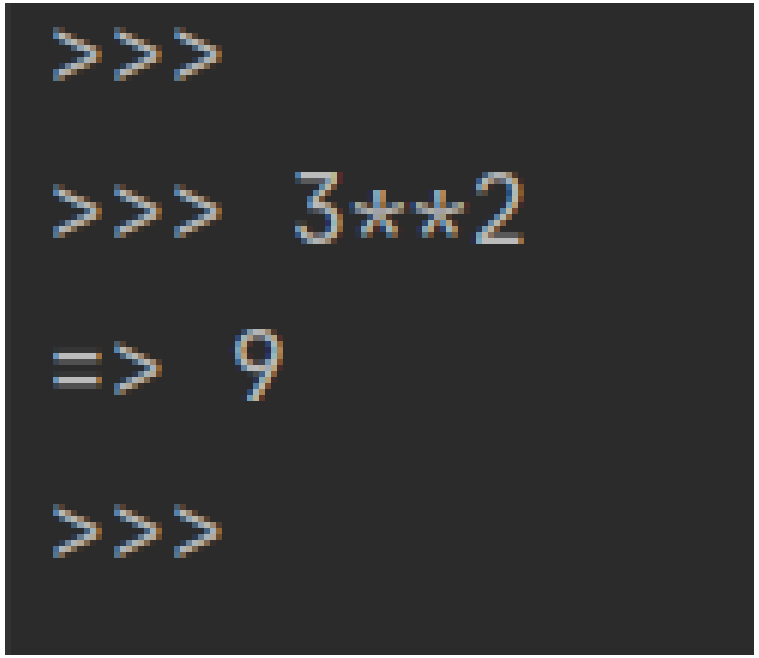
Abbiamo a disposizione anche **l'operatore modulo**, il quale **restituisce il resto di una divisione**
L'operatore modulo è **spesso utilizzato per determinare se un valore è pari o dispari.**
Nel caso riportato a destra la condizione stamperà a schermo 5 è dispari dato che il resto dell'operazione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

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

<?php

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

Ciclo foreach in PHP

Il foreach, permette di **ciclare** 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

password_hash()

La funzione password_hash() crea un nuovo hash della password utilizzando un potente algoritmo di hashing unidirezionale. password_hash() è compatibile con crypt(). Pertanto, gli hash delle password creati da crypt() possono essere utilizzati con password_hash().

Metodi di crypt:
PASSWORD_DEFAULT
PASSWORD_BCRYPT
PASSWORD_ARGON2I
PASSWORD_ARGON2ID

```php
<?php
 echo password_hash("test_password", PASSWORD_DEFAULT);
?>

//
$2y$10$EcdwlgmmKqz0lrrgAX8AF.8/SkLdYGVrTNRfAg3OY1Swqa6VF.Qh2
```

ogni volta che si esegue l'hash cambierà

```php
print password_hash("miaPwd", PASSWORD_DEFAULT);
print PHP_EOL;
> $2y$10$UAg.1WX5QEShW4fT2JStBO.98BWRc..RdHBPESNDFdyy6Eaa1a2MS
print password_hash("miaPwd", PASSWORD_BCRYPT);
print PHP_EOL;
> $2y$10$EcZXgvxAjoPmYW6ufK/RIOIfkL.XD7Hmr1J3DAu.oF.yuT8xX9MdC
```

crypt()

consente di criptare una stringa utilizzando una chiave di criptazione

```
print "crypt:" . crypt("mia_password",
'mia_$chiave_stringa' );


$user_input='12+#a345';
$pass = urlencode($user_input);
$pass_crypt = crypt($pass, "");
if
($pass_crypt==crypt($pass,$pass_crypt)
) {
    echo"Success! Valid password";
} else {
    echo"Invalid password";
}


?>
```

Output

Standard DES: stqAdD7zlbByI

```
// Set the password
$password = 'mypassword';

// Get the hash, letting the
salt be automatically generat
ed; not recommended
$hash = crypt($password);
?>
```

explode('chr', $stringa) : array- Suddividere una stringa

Spesso capita di dover **separare una stringa in più parti in base ad un carattere separatore**, ad esempio lo / nella data vista in uno degli esempi precedenti.

In questo caso possiamo **usa**re la funzione **explode()** che prende in ingresso 3 parametri:

Parametro	Descrizione
$separator	Carattere separatore.
$string	Stringa da dividere.
$limit	Opzionale, numero massimo di elementi da estrarre.

Esempio

```
$data = '01/02/2016';
$elementi = explode('/', $data);
   // conterrà
   // Array
   // (
   //  [0] => 01
   //  [1] => 02
   //  [2] => 2016
   // )
```

implode('chr',array) convertire un array in una stringa

La funzione con cui possiamo **convertire un array in una stringa** si chiama **implode()** che prende in ingresso 2 parametri:

Parametro	Descrizione
$glue	Il carattere separatore.
$pieces	L'array da unire.

```php
print implode(
    '-',
    [
        0 => 'primo',
        1 => 'secondo',
        2 => 'terzo',
    ]
);
/*
primo-secondo-terzo
 */
```

Modificare le maiuscole/minuscole

PHP mette a disposizione un **set di funzioni per modificare le maiuscole/minuscole** a seconda dell'esigenza:

Parametro	Descrizione
strtoupper()	Per convertire la stringa tutta in maiuscolo.
strtolower()	Per convertire la stringa tutta in minuscolo.
lcfirst()	Per convertire la prima lettera della stringa in minuscolo.
ucfirst()	Per convertire la prima lettera della stringa in maiuscolo.
ucwords()	Per convertire la prima lettera di tutte le parole in maiuscolo.

Esempi:

```
$stringa = 'questa stringa contiene tutti caratteri minuscoli';
echo strtoupper($stringa); // stamperà QUESTA STRINGA CONTIENE TUTTI CARATTERI MINUSCOLI
echo ucfirst($stringa); // stamperà Questa stringa contiene tutti caratteri minuscoli
echo ucwords($stringa); // stamperà Questa Stringa Contiene Tutti Caratteri Minuscoli
```

Rimuovere spazi all'inizio o alla fine di una stringa

Spesso quando riceviamo delle stringhe in input abbiamo bisogno di **ripulirle da spazi o altri caratteri in eccesso ad inizio o fine stringa.**

Le funzioni a disposizione sono:

Parametro	Descrizione
trim()	Per rimuovere i caratteri sia all'inizio che alla fine.
ltrim()	Per rimuovere i caratteri solo a sinistra.
rtrim()	Per rimuovere i caratteri solo a destra.

Esempi

```
$stringa = ' ciao ';
echo trim($stringa);  // stamperà 'ciao' senza spazi iniziali e finali
echo ltrim($stringa); // stamperà 'ciao ' senza spazi iniziali
echo rtrim($stringa); // stamperà ' ciao' senza spazi finali
```

sprintf()

Si possono creare delle stringe concatenate grazie agli operatori.

In alcuni casi però che la concatenazione è piuttosto scomoda da gestire.

Inoltre si ha lo svantaggio di aumentare molto la probabilità di commettere errori, come per esempio gli spazi dimenticati tra le stringhe e le variabili.
Per risolvere questo problema PHP ci mette a disposizione una funzione molto interessante: sprintf().

Attraverso di essa possiamo generare delle stringhe formattate in maniera più semplice e scrivendo del codice più leggibile

```
$nome = 'Simone';
$eta = 29;
$citta = 'Forli';
$provincia = 'Forli-Cesena';
$regione = 'Emilia-Romagna';
$formato = '%s ha %d anni ed abita a %s in provincia di %s, nella regione %s';
$stringa = sprintf($formato, $nome, $eta, $citta, $provincia, $regione);
```

I formati che abbiamo a disposizione:

Formato	Descrizione
%%	Segno di percentuale.
%b	Numero binario.
%c	Carattere in formato ASCII.
%d	Decimale con segno.
%e	Notazione scientifica in minuscolo (e.g. 1.2e+2).
%E	Notazione scientifica in maiuscolo (e.g. 1.2E+2).
%u	Decimale senza segno.
%f	Numero in virgola mobile (con verifica delle impostazioni di sistema).
%F	Numero in virgola mobile (senza verifica delle impostazioni di sistema).
%o	Numero in formato ottale.
%s	Stringa.
%x	Numero in formato esadecimale (lettere in minuscolo).
%X	Numero in formato esadecimale (lettere in maiuscolo).

I formati più utilizzati nello sviluppo quotidiano sono sicuramente %d %f %s

```
$nome = 'TuoNone';
$quantita = 15;
$sconto = 10;
$totale = 125.9999;
$formato = "%s ha ordinato %d pezzi con uno sconto del %d%% per un totale di euro %.2f.";
$stringa = sprintf($formato, $nome, $quantita, $sconto, $totale);
```

%.2f

Osservando con attenzione la stringa contenente il formato di stampa noteremo alcune particolarità:

- **%%** è utilizzato per **stampare un simbolo della percentuale;**
- **%.2f** è utilizzato per **formattare il float** a 2 decimali

la funzione ha anche effettuato l'arrotondamento automaticamente.

```
$nome = 'TuoNome';
$quantita = 15;
$sconto = 10;
$totale = 125.9999;
$formato = "%s ha ordinato %d pezzi con uno sconto del %d%% per un totale di euro %.2f.";
$stringa = sprintf($formato, $nome, $quantita, $sconto, $totale);
```

Simone ha ordinato 15 pezzi con uno sconto del 10% per un totale di euro 126.00

printf()

Se volessimo stampare direttamente a schermo il risultato potremmo utilizzare la funzione printf() che ha **lo stesso funzionamento di sprintf() e, in aggiunta, mostra a schermo la stringa formattata.**

```
printf($formato, $nome, $quantita, $sconto, $totale);
```

Confronto tra stringhe in PHP == / ===

Attraverso l'operatore == è possibile verificare se due stringhe sono uguali tra loro.

I confronti tra stringhe in PHP sono case sensitive (letteralmente “sensibili alle maiuscole “).

```
$stringa = "stringa";
if ($stringa == "stringa") {
    echo "Le stringhe sono uguali.";
} else {
    echo "Le stringhe non coincidono.";
}
```

html_entity_decode()

La funzione html_entity_decode() converte le entità HTML in caratteri.
La funzione html_entity_decode() è l'opposto di htmlentities() .

```
print html_entity_decode("da &lt; test &gt;");
#da < test >
```

```
<?php
$str = '&lt;a
href=&quot;https://www.school.com&quot;&gt;scho
ol.com&lt;/a&gt;';
echo html_entity_decode($str);
?>
```

L'output HTML del codice sopra sarà (Visualizza sorgente):
school.com

L'output del browser del codice sopra sarà:
school.com

htmlentities()

converte i caratteri in entità HTML.

```
print htmlentities("<div>");

//&lt;div&gt;
```

<?php
$str = 'Go to school.com';
echo htmlentities($str);
?>
L'output HTML del codice sopra sarà (Visualizza sorgente):
Go to school.com
L'output del browser del codice sopra sarà:
Go to school.com

md5()

La funzione md5() calcola l'hash MD5 di una stringa.
La funzione md5() utilizza l'algoritmo MD5 Message-Digest di RSA Data Security, Inc..

Da RFC 1321 - L'algoritmo MD5 Message-Digest: "L' algoritmo MD5 message-digest prende come input un messaggio di lunghezza arbitraria e produce come output una "impronta digitale" o "message digest" a 128 bit dell'input. L'algoritmo MD5 è destinato alle applicazioni di firma digitale, in cui un file di grandi dimensioni deve essere "compresso" in modo sicuro prima di essere crittografato con una chiave privata (segreta) in un sistema crittografico a chiave pubblica come RSA."
Per calcolare l'hash MD5 di un file, utilizzare la funzione md5_file() .

```php
<?php
$str = "Ciao La mia stringa";
echo md5($str);
?>
```

nl2br()

La funzione nl2br() inserisce interruzioni di riga HTML (
 o
) davanti a ogni nuova riga (\n) in una stringa.

```php
<?php
$str = "Ciao \nLa mia\n
stringa";
echo nl2br($str);

/*
Ciao <br />
La mia<br />
 stringa
 */
```

sha1() 160bit hash

La funzione sha1() calcola l'hash SHA-1 di una stringa.
La funzione sha1() utilizza l'algoritmo US Secure Hash 1.
per calcolare l'hash SHA-1 di un file, utilizzare la funzione sha1_file() .

più sicuro di MD5

```php
<?php
$str = "mia#password";
echo sha1($str);

/*
e6cc206d15f0ca02bcff6b36cfef2dcfddd04310
 */
```

parse_str() – per querystring

La funzione parse_str() analizza una querystring e restituisce un array

DEPRECATED as of PHP 7.2.

```php
<?php
/**
 * fare dei test utilizzando
parse_str
 */

parse_str("p=password&nome=test",
$ar);
print_r($ar);


/*
Array
(
    [p] => password
    [nome] => test
)
 */
```

setlocale()

La funzione setlocale() imposta le informazioni sulla localizzazione.
Le informazioni locali sono la lingua, la moneta, l'ora e altre informazioni specifiche per un'area geografica.
Nota: la funzione setlocale() cambia la localizzazione solo per lo script corrente.
Suggerimento: le informazioni sulla localizzazione possono essere impostate sui valori predefiniti del sistema con setlocale(LC_ALL,NULL)

```
setlocale(int $category, string $locales, string ...$rest): string|false
```

$category:
- LC_ALL for all of the below
- LC_COLLATE for string comparison, see strcoll()
- LC_CTYPE for character classification and conversion, for example strtoupper()
- LC_MONETARY for localeconv()
- LC_NUMERIC for decimal separator (See also localeconv())
- LC_TIME for date and time formatting with strftime()
- LC_MESSAGES for system responses (available if PHP was compiled with libintl)

```
$loc=setlocale(LC_ALL,
'de_DE@euro', 'de_DE',
'deu_deu');
echo strftime('%B');
//Februar
```

```
$loc=setlocale(LC_ALL,
'it_IT@euro', 'it_IT',
'ita_ita');
echo strftime('%B');
//febbraio
```

str_pad ()

La funzione str_pad() aggiunge un carattere / serie di caratteri ripetuti in una nuova stringa

di default li aggiunge a destra:
STR_PAD_RIGHT
possibili parametri:
STR_PAD_LEFT
STR_PAD_BOTH

```
<?php
$input = "Film";
echo str_pad($input, 10);
// produces "Film      "
echo str_pad($input, 10, "-=", STR_PAD_LEFT);
// produces "-=-=-Film"
echo str_pad($input, 10, "_", STR_PAD_BOTH);
// produces "__Film___"
echo str_pad($input,  6, "___");
// produces "Film_"
echo str_pad($input,  3, "*");
// produces "Film"
```

str_repeat()

La funzione **str_repeat()** ripete una stringa un numero specificato di volte.

```
<?php
echo str_repeat("Wow",13);
?>
```

Output:
WowWowWowWowWowWowWowWowWowWowWowWowWow

str_split()

La funzione str_split() divide una stringa in un array.

Valore di ritorno:

Se length è minore di 1, la funzione str_split() restituirà FALSE.

Se length è maggiore della lunghezza della stringa, l'intera stringa verrà restituita come unico elemento dell'array.

```
<?php
print_r(str_split("Hello"));
?>
```

Risultato:

Array ([0] => H [1] => e [2] => l [3] => l [4] => o)

```php
<?php
print_r(str_split("Hello",2));
// Array
// (
//     [0] => He
//     [1] => ll
//     [2] => o
// )
```

strcmp()

La funzione strcmp() confronta due stringhe.
Nota: la funzione strcmp() è binary-safe e fa distinzione tra maiuscole e minuscole.

Suggerimento: questa funzione è simile alla funzione strncmp() , con la differenza che è possibile specificare il numero di caratteri di ciascuna stringa da utilizzare nel confronto con strncmp().
Valore di ritorno:
Questa funzione **restituisce:**
0 - se le due stringhe sono **uguali**
<0 - se **stringa1 è minore di stringa2**
>0 - se **stringa1 è maggiore di stringa2**

```
<?php
echo strcmp("Hello world!","Hello world!");
?>
```

Output:
0
Se questa funzione restituisce 0, le due stringhe sono uguali.

strcasecmp()

Nel caso in cui abbiamo bisogno di effettuare **confronti non sensibili alle maiuscole** possiamo utilizzare la funzione strcasecmp()
Ha un comportamento del tutto identico a strcmp senza però essere case-sensitive.

Utilizzando strcasecmp() l'esempio ci restituirà il messaggio “Le stringhe sono uguali” in quanto la funzione è case-insensitive.

```
$stringa = "Stringa";
if (strcasecmp($stringa, "stringa") == 0) {
    echo "Le stringhe sono uguali";
} else {
    echo "Le stringhe non coincidono";
}
```

strstr()

La funzione strstr() cerca la prima occorrenza di una stringa all'interno di un'altra stringa.
Nota: questa funzione è a binary safe.
Nota: questa funzione distingue tra maiuscole e minuscole.
Per una ricerca senza distinzione tra maiuscole e minuscole, utilizzare la funzione stristr() .

binary safe: Significa che la funzione funzionerà correttamente quando si passano dati binari arbitrari (cioè stringhe contenenti byte non ASCII e / o byte nulli).
esempio

$str = "abc\x00abc";
echo strlen($str); //gives 7, not 3!

\x00 = null byte per la terminazione della stringa

```php
$email  = 'name@example.com';
$domain = strstr($email, '@');
echo $domain; // prints @example.com

$user = strstr($email, '@', true);
echo $user; // prints name
```

strip_tags()

La funzione strip_tags() **rimuove i tag da una stringa** HTML, XML e PHP.
Nota: i commenti HTML vengono sempre eliminati. Questo non può essere modificato con il parametro allow.
Nota: questa funzione è a sicurezza binaria.

```
<?php
echo strip_tags("Hello <b><i>world!</i></b>","<b>");
?>
```

Output:
Hello **world**!

```
// Allow <p> and <a>
echo strip_tags($text, '<p><a>');
in questo caso saranno consentiti I tag p e a

print strip_tags("<p>ciao");
// ciao
```

stripslashes()

La funzione stripslashes() rimuove le barre rovesciate aggiunte dalla funzione addslashes().

Suggerimento: questa funzione può essere utilizzata per ripulire i dati recuperati da un database o da un modulo HTML.

```
<?php
echo stripslashes("Hello \World!");
?>
```

Output:

Hello World!

stripos ()

La funzione stripos() trova la posizione della prima occorrenza di una stringa all'interno di un'altra stringa.
Nota: la funzione stripos() non fa distinzione tra maiuscole e minuscole. (CASE INSENSITIVE)
Nota: questa funzione è a sicurezza binaria.

Funzioni correlate:

strripos() - Trova la posizione dell'ultima occorrenza DESTRA di una stringa all'interno di un'altra stringa (senza distinzione tra maiuscole e minuscole) (CASE INSENSITIVE)

strpos() - Trova la posizione della prima occorrenza SINISTRA di una stringa all'interno di un'altra stringa (con distinzione tra maiuscole e minuscole) (CASE SENSITIVE)

strrpos() - Trova la posizione dell'ultima occorrenza DESTRA di una stringa all'interno di un'altra stringa (con distinzione tra maiuscole e minuscole) (CASE SENSITIVE)

```php
<?php

print stripos("questa è una
frase di prova", "f");
// 14

print stripos("questa è una
frase di prova", "F");
// 14
```

Alcune funzioni Math

The pi() function returns the value of PI:
\

The min() and max() functions can be used to find the lowest or highest value in a list of arguments:

```
<?php
echo(min(0, 150, 30, 20, -8, -200));  // returns -200
echo(max(0, 150, 30, 20, -8, -200));  // returns 150
?>
```

The abs() function returns the absolute (positive) value of a number:

```
<?php
echo(abs(-6.7));  // returns 6.7
?>
```

The sqrt() function returns the square root of a number:

```
<?php
echo(sqrt(64));  // returns 8
?>
```

The round() function rounds a floating-point number to its nearest **integer**:

```
<?php
echo(round(0.60));  // returns 1
echo(round(0.49));  // returns 0
 var_dump(round(5.045, 2)); //5.05
?>
```

The rand() function generates a random number:

```
<?php
 print rand(10,500); //462

 print rand();//427379992
```

FUNZIONI ANONIME

La funzioni lambda (o “funzioni anonime”) sono semplicemente funzioni usa e getta, che possono essere definite in qualsiasi momento e che sono in genere associato ad una variabile.

```
$mia_func = function ($v) {
    print ($v * 2);
};
$mia_func(5); //10
```

FUNZIONI ANONIME

possiamo creare funzioni anonime che possono essere utilizzate in casi particolari dove, ad esempio, la funzione viene passata come parametro ad un'altra funzione.
È sconsigliabile utilizzarla **nel caso di funzioni da richiamare in più punti** della nostra applicazione, non avendo un nome sarebbe impossibile richiamarla.

```php
//array_map – Applies the callback to the elements of the given arrays
$array = [2, 4, 6];
$array2 = array_map(function($val) {
    return $val * 2;
}, $array);
print_r($array2);
```

```
//Array
(
    [0] => 4
    [1] => 8
    [2] => 12
)
```

CLOSURE

Quando una funzione è dichiarata, ha la capacità di fare riferimento a tutte le variabili che sono dichiarate nel suo ambito. Una variabile dichiarata al di fuori della funzione non sarà quindi "visibile" al suo interno.
Le closure, in parole povere, non sono altro che funzioni anonime che conoscono alcune variabili che non sono state definite al loro interno.

La variabile di default viene importata per valore, ciò significa che se aggiorniamo il valore della variabile importata nella closure, la variabile esterna non sarà aggiornata.

```php
$saluto = 'Ciao Mondo';
$closure = function() use($saluto) {
   return $saluto;
};

echo $closure(); // Ciao Mondo
```

```php
<?php
//array_map – Applies the callback to the
elements of the given arrays
$array = [2, 4, 6];
$coeff=4;
array_map(function($val) use($coeff) {
    echo ($val * $coeff)."\n";
}, $array);
```

CLOSURE binding

dal PHP 7 le closure hanno integrato il metodo call che consente di passare dei dati di ambiente alle closure

```php
<?php
class A {
    private $hi = 'Ciao';
}

$closure = function () {
    return $this->hi;
};
print $closure->call(new A());
```

PSR-12

Closures MUST be declared with a space after the function keyword, and a space before and after the use keyword.

```
$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // body
};
```

Tipizzazione

Tipi in ingresso
Il costrutto declare viene utilizzato per modificare il comportamento di alcune direttive del linguaggio in fase di esecuzione del codice. La sintassi di declare è simile a quella di altre
declare(strict_types=1);

Per specificare il tipo in ingresso di una funzione, è necessario aggiungere il nome del tipo prima del nome del parametro come mostra il seguente esempio

```
<?php
declare(strict_types=1); // attiviamo il controllo sui tipi

function printHello(string $name) // la funzione accetta solo valori di tipo string
{
   echo "Hello $name";
}

printHello("Test"); //Hello Test
printHello(4); // genera un errore
PHP Fatal error:  Uncaught TypeError: Argument 1 passed to printHello() must be of the type string, int given...
```

Tipi in uscita
specificare il tipo del valore ritornato da una funzione. Per specificare il tipo in uscita delle funzioni è necessario aggiungere il carattere dei due punti : dopo la parentesi di chiusura della funzione

```php
<?php
declare(strict_types=1);

function sum(int $a, int $b): int
{
    return $a + $b;
}

echo sum(4, 9);

function sum2(int $a, int $b): int
{
    return 4.4 + $b;
}

echo sum2(4, 9); //PHP Fatal error: Uncaught TypeError: Return value of sum() must be of the type int, float returned...
```

Scope delle variabili

Per **scope** si intende la visibilità di una variabile, ovvero la "**porzione**" di codice in cui essa è accessibile.

Immaginiamo di definire una variabile $a e di volerla utilizzare in una funzione. Richiamando direttamente $a all'interno della funzione essa non avrà valore dato.
restituirà 10 e solleverà un Notice come il seguente:
PHP Notice: Undefined variable: a in php shell code on line 2
Questo perché nel contesto della funzione $a non è stata definita.

```
$a = 10;
function sum($b) {
    return $a + $b;
}
echo sum(10);
//PHP Notice:  Undefined variable: a in php shell code on line 2
```

Come dovrebbe essere:

```
$a = 10;
function sum($a, $b) {
    return $a + $b;
}
echo sum($a, 10); //in questo caso il risultato sarà 20
```

Usare la keyword global

Un'altra soluzione è **dichiarare la variabile come globale così da poterla utilizzare anche all'interno di altri scope.**
Con la keyword global possiamo indicare a PHP che ogni riferimento alla variabile $a deve essere cercato nello scope globale e non in quello locale.

Variabile non definita //esempio precedente errato

```
$a = 10;
function sum($b) {
    return $a + $b;
}
echo sum(10);
```

Invece Con la Keyword global

```
$a = 10;
function sum($b) {
    global $a;
    return $a + $b;
}
echo sum(10); //anche in questo caso il risultato sarà 20
```

Accedere all'array $GLOBALS

L'array $GLOBALS contiene tutte le variabili dichiarate fino a quel momento comprese le variabili predefinite ($_GET, $_POST, $_SERVER..). Possiamo quindi utilizzarlo per accedere al valore della variabile.

Salvo in rari casi è sconsigliato utilizzare questo approccio.

Quando possibile è consigliabile invece strutturare il codice affinché esso abbia un proprio scope, così da evitare risultati indesiderati.

```
$a = 10;
function sum($b) {
    return $GLOBALS['a'] + $b;
}
echo sum(10); //anche in questo caso il risultato sarà 20
```

Keyword global e array $GLOBALS .

```
$a = 10;
function sum($b) {
    global $a;
    return $a + $b;
}
echo sum(10); //anche in questo caso il
risultato sarà 20
```

```
$a = 10;
function sum($b) {
    return $GLOBALS['a'] + $b;
}
echo sum(10); //anche in questo caso il
risultato sarà 20
```

Includere file nel codice

È possibile **inserire il contenuto di un file PHP in un altro file** PHP **con i costrutti** include e require.
Per includere il file è sufficiente richiamarlo con il path completo.
Il risultato è sostanzialmente identico tranne per come vengono gestiti gli errori.
Se si verifica un errore all'interno del codice incluso **con require l'esecuzione viene arrestata con un fatal error E_COMPILE_ERROR**.
Nel caso di errore **con include l'esecuzione dello script prosegue e viene sollevato un E_WARNING**.

```
include 'nomedelfile.php';
//oppure
require 'nomedelfile.php';
```

Scope delle variabili con include e require

Una variabile definita prima di includere un file è sempre visibile all'interno di un file incluso.

Supponiamo di creare un file chiamato moltiplica.php con contenuto:

```
<?php
   $a = $a * 100;
?>
```

e di avere il nostro codice principale in un file index.php, nella stessa cartella dell'altro file, con contenuto:

```
<?php
   $a = 10;
   include 'moltiplica.php';
   echo $a;
?>
```

Eseguendo il codice di index.php verrà stampato il valore 1000 perché la variabile $a è visibile all'interno del file incluso.

include_once / require_once
Evitare inclusioni ripetute dello stesso file

Supponiamo di avere un codice in cui in base a determinate condizioni potrebbe capitare di includere più volte lo stesso file. Se nel file incluso vengono definite funzioni o modificati i valori di variabili, sicuramente si possono avere risultati inaspettati.

Per evitare tali situazioni esistono include_once e require_once.
Entrambi consentono di includere una sola volta lo stesso file anche se il costrutto venisse richiamato più volte.

Riutilizzando moltiplica.php immaginiamo di avere un index.php come il seguente:

```
<?php
    $a = 10;
    include 'moltiplica.php';
    include 'moltiplica.php';
    echo $a;
?>
```

Il risultato non sarà 1000 ma 100000 perché la moltiplicazione viene eseguita due volte.
Modificando il codice con il costrutto appena introdotto, invece, il risultato continuerà ad essere 1000:

```
<?php
    $a = 10;
    include_once 'moltiplica.php';
    include_once 'moltiplica.php';
    echo $a;
?>
```

empty()

La funzione empty() controlla se una variabile è vuota o meno.
Questa funzione restituisce true se la variabile non esiste oppure è vuota, altrimenti restituisce false.

per i seguenti valori restituisce false:
0
0.0
"0"
""
NULLO
FALSO
Array()

```
>>> empty($a)
=> true
>>> $a=0;
=> 0
>>> empty($a)
=> true
>>> $a=1;
=> 1
>>> empty($a)
=> false
>>>
```

floatval () – converte stringa in float

La funzione floatval() restituisce il valore float di una variabile.

Il valore float della variabile in caso di successo,
0 in caso di fallimento.

Un array vuoto restituirà 0 e un array non vuoto restituirà 1

```
<?php
$a = "1234.56789";
echo floatval($a) . "<br>";

$b = "1234.56789Hello";
echo floatval($b) . "<br>";

$c = "Hello1234.56789";
echo floatval($c) . "<br>";
?>
```

Output:
1234.56789
1234.56789
0

gettype ()

La funzione gettype() restituisce il tipo di una variabile.

Valore di ritorno: Il tipo come stringa.

Può essere **uno dei seguenti** valori:

- "boolean",
- "integer",
- "double",
- "string",
- "array",
- "object",
- "NULL",
- "unknown type"

```
<?php
$a = 3;
echo gettype($a) . "<br>";

$b = 3.2;
echo gettype($b) . "<br>";

$c = "Hello";
echo gettype($c) . "<br>";

$d = array();
echo gettype($d) . "<br>";

$e = array("red", "green", "blue");
echo gettype($e) . "<br>";

$f = NULL;
echo gettype($f) . "<br>";

$g = false;
echo gettype($g) . "<br>";
?>
```

Output:

```
Integer

double

string

array

array

NULL

boolean
```

Funzioni di gestione delle variabili // intval ()

La funzione intval() restituisce il valore intero di una variabile.
Il valore intero della variabile in caso di successo,
0 in caso di fallimento.
Un array vuoto restituirà 0 e un array non vuoto restituirà 1

```
<?php
$a = 32;
echo intval($a) . "<br>";
$b = 3.2;
echo intval($b) . "<br>";
$c = "32.5";
echo intval($c) . "<br>";
$d = array();
echo intval($d) . "<br>";
$e = array("red", "green",
"blue");
echo intval($e) . "<br>";
?>
```

Output:

```
32
3
32
0
1
```

is_array()

La funzione **is_array() controlla se una variabile è un array** o meno.
Questa funzione restituisce true (1) se la variabile è un array, altrimenti restituisce false/nothing.

```
<?php
$a = "Hello";
echo "a is " . is_array($a) . "<br>";
$b = array("red", "green", "blue");
echo "b is " . is_array($b) . "<br>";
$c = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");
echo "c is " . is_array($c) . "<br>";
$d = "red, green, blue";
echo "d is " . is_array($d) . "<br>";
?>
```

is_bool()

La funzione **is_bool() controlla se una variabile è booleana o meno.**
Questa funzione restituisce true (1) se la variabile è booleana, altrimenti restituisce false/nothing.

```
<?php
$a = 1;
echo "a is " . is_bool($a) . "<br>";
$b = 0;
echo "b is " . is_bool($b) . "<br>";
$c = true;
echo "c is " . is_bool($c) . "<br>";
$d = false;
echo "d is " . is_bool($d) . "<br>";
?>
```

is_double ()

La funzione **is_double() controlla se una variabile è di tipo float** o meno.
Questa funzione è un alias di is_float() .

```
<?php
$a = 32;
echo "a is " . is_double($a) . "<br>";
$b = 0;
echo "b is " . is_double($b) . "<br>";
$c = 32.5;
echo "c is " . is_double($c) . "<br>";
$d = "32";
echo "d is " . is_double($d) . "<br>";

$e = true;
echo "e is " . is_double($e) . "<br>";
$f = "null";
echo "f is " . is_double($f) . "<br>";
$g = 1.e3;
echo "g is " . is_double($g) . "<br>";
?>
```

is_float()

La funzione **is_float() controlla se una variabile è di tipo float o meno**.
Questa funzione restituisce true (1) se la variabile è di tipo float, altrimenti restituisce false.

```php
<?php
$a = 32;
echo "a is " . is_float($a) . "<br>";
$b = 0;
echo "b is " . is_float($b) . "<br>";
$c = 32.5;
echo "c is " . is_float($c) . "<br>";
$d = "32";
echo "d is " . is_float($d) . "<br>";
$e = true;
echo "e is " . is_float($e) . "<br>";
$f = "null";
echo "f is " . is_float($f) . "<br>";
$g = 1.e3;
echo "g is " . is_float($g) . "<br>";
?>
```

is_int()

La funzione **is_int() controlla se una variabile è di tipo intero o meno.**
Questa funzione restituisce true (1) se la variabile è di tipo intero, altrimenti restituisce false.

```php
<?php
$a = 32;
echo "a is " . is_int($a) . "<br>";
$b = 0;
echo "b is " . is_int($b) . "<br>";
$c = 32.5;
echo "c is " . is_int($c) . "<br>";
$d = "32";
echo "d is " . is_int($d) . "<br>";
$e = true;
echo "e is " . is_int($e) . "<br>";
$f = "null";
echo "f is " . is_int($f) . "<br>";
?>
```

is_integer()

La funzione **is_integer() controlla se una variabile è di tipo intero o meno.**
Questa funzione è un alias di is_int() .

```
<?php
$a = 32;
echo "a is " . is_integer($a) . "<br>";
$b = 0;
echo "b is " . is_integer($b) . "<br>";
$c = 32.5;
echo "c is " . is_integer($c) . "<br>";
$d = "32";
echo "d is " . is_integer($d) . "<br>";
$e = true;
echo "e is " . is_integer($e) . "<br>";
$f = "null";
echo "f is " . is_integer($f) . "<br>";
?>
```

is_null()

La funzione **is_null() controlla se una variabile è NULL o meno.**
Questa funzione restituisce true (1) se la variabile è NULL, altrimenti restituisce false/nothing.

```
<?php
$a = 0;
echo "a is " . is_null($a) . "<br>";
$b = null;
echo "b is " . is_null($b) . "<br>";
$c = "null";
echo "c is " . is_null($c) . "<br>";
$d = NULL;
echo "d is " . is_null($d) . "<br>";
?>
```

is_numeric()

La funzione **is_numeric() controlla se una variabile è un numero o una stringa numerica**.
Questa funzione restituisce true (1) se la variabile è un numero o una stringa numerica, altrimenti restituisce false/nothing.

```php
<?php
$a = 32;
echo "a is " . is_numeric($a) . "<br>";
$b = 0;
echo "b is " . is_numeric($b) . "<br>";
$c = 32.5;
echo "c is " . is_numeric($c) . "<br>";
$d = "32";
echo "d is " . is_numeric($d) . "<br>";
$e = true;
echo "e is " . is_numeric($e) . "<br>";
$f = null;
echo "f is " . is_numeric($f) . "<br>";
?>
```

is_object()

La funzione **is_object() controlla se una variabile è un oggetto.**
Questa funzione restituisce true (1) se la variabile è un oggetto, altrimenti restituisce false/nothing.

```
<?php
function get_cars($obj) {
  if (!is_object($obj)) {
    return false;
  }
return $obj->cars;
}
$obj = new stdClass();
$obj->cars = array("Volvo", "BMW",
"Audi");
var_dump(get_cars(null));
echo "<br>";
var_dump(get_cars($obj));
?>
```

is_string()

La funzione **is_string() controlla se una variabile è di tipo string o meno**.
Questa funzione restituisce true (1) se la variabile è di tipo string, altrimenti restituisce false/nothing.

```php
<?php
$a = "Hello";
echo "a is " . is_string($a) . "<br>";
$b = 0;
echo "b is " . is_string($b) . "<br>";
$c = 32;
echo "c is " . is_string($c) . "<br>";
$d = "32";
echo "d is " . is_string($d) . "<br>";
$e = true;
echo "e is " . is_string($e) . "<br>";
$f = "null";
echo "f is " . is_string($f) . "<br>";
$g = "";
echo "g is " . is_string($g) . "<br>";
?>
```

is_scalar()

La funzione **is_scalar() controlla se una variabile è scalare o meno.**

Questa funzione restituisce true (1) se la variabile è scalare, altrimenti restituisce false/nothing.

Interi, float, stringhe o boolean possono essere variabili scalari.

Matrici, oggetti e risorse non lo sono.

```php
<?php
$a = "Hello";
echo "a is " . is_scalar($a) . "<br>";
$b = 0;
echo "b is " . is_scalar($b) . "<br>";
$c = 32;
echo "c is " . is_scalar($c) . "<br>";
$d = NULL;
echo "d is " . is_scalar($d) . "<br>";
$e = array("red", "green", "blue");
echo "e is " . is_scalar($e) . "<br>";
?>
```

isset()

La funzione **isset() controlla se una variabile è impostata**, il che significa che deve essere dichiarata e non è NULL.
Questa funzione **restituisce true se la variabile esiste e non è NULL, altrimenti restituisce false.**
Nota: se vengono fornite più variabili, questa funzione restituirà true solo se tutte le variabili sono impostate.
Suggerimento: una variabile può essere annullata con la funzione unset().

```
<?php
$a = 0;
// True because $a is set
if (isset($a)) {
  echo "Variable 'a' is set.<br>";
}
$b = null;
// False because $b is NULL
if (isset($b)) {
  echo "Variable 'b' is set.";
}
?>
```

print_r()

La funzione **print_r() stampa le informazioni su una variabile in un modo più leggibile.**

Valore di ritorno:

Se la variabile è intera, float o stringa, verrà stampato il valore stesso.

Se la variabile è una matrice o un oggetto, questa funzione restituisce chiavi ed elementi.

Se il parametro di ritorno è impostato su TRUE, questa funzione restituisce una stringa

```
<?php
$a = array("red", "green", "blue");
print_r($a);
echo "<br>";
$b = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
print_r($b);
?>
```

Risultato:

```
Array ( [0] => red [1] => green [2] => blue )
Array ( [Peter] => 35 [Ben] => 37 [Joe] => 43 )
```

serialize()

La funzione **serialize() converte una rappresentazione memorizzabile di un valore**. Serializzare i dati significa **convertire un valore in una sequenza di bit**, in modo che possa essere archiviato in un file, un buffer di memoria o trasmesso attraverso una rete.

```
<?php
$data = serialize(array("Red", "Green", "Blue"));
echo $data;
?>
```

Risultato:

```
a:3:{i:0;s:3:"Red";i:1;s:5:"Green";i:2;s:4:"Blue";}
```

settype ()

La funzione **settype() converte una variabile in un tipo specifico.**
Valore di ritorno:
VERO in caso di successo,
FALSO in caso di fallimento

```
<?php
$a = "32"; // string
settype($a, "integer"); // $a is now integer
$b = 32; // integer
settype($b, "string"); // $b is now string
$c = true; // boolean
settype($c, "integer"); // $c is now integer (1)
?>
```

strval()

La funzione strval() restituisce il valore stringa di una variabile.

es traducendo un oggetto viene chiamato il magic method __toString

```php
<?php
class MiaClasse {
    private $hi = 'Ciao';
    public function __toString()
    {
        return __CLASS__;
    }
}

print strval(new MiaClasse);
```

```php
<?php
$a = "Hello";
echo strval($a) . "<br>";
$b = "1234.56789";
echo strval($b) . "<br>";
$c = "1234.56789Hello";
echo strval($c) . "<br>";
$d = "Hello1234.56789";
echo strval($d) . "<br>";
$e = 1234;
echo strval($e) . "<br>";
?>
/*
Hello
1234.56789
1234.56789Hello
Hello1234.56789
1234
*/
```

unserialize()

La funzione **unserialize() converte i dati serializzati in dati effettivi.**
Valore di ritorno:
Il valore convertito può essere un valore booleano, intero, float, stringa, array o oggetto.
FALSE e un E_NOTICE in caso di errore

```
<?php
$data = serialize(array("Red", "Green", "Blue"));
echo $data . "<br>";

$test = unserialize($data);
var_dump($test);
?>
```

Risultato:

```
a:3:{i:0;s:3:"Red";i:1;s:5:"Green";i:2;s:4:"Blue";}
array(3) { [0]=> string(3) "Red" [1]=> string(5)
"Green" [2]=> string(4) "Blue" }
```

unset()

La funzione **unset() annulla l'impostazione di una variabile.**

caso di unset e array:

```
$ar = array(
    "primo" => 123,
    "secondo" => 321,
);
unset ($ar["primo"]);
print_r ($ar);
/*
Array
(
    [secondo] => 321
)
 */
```

```
<?php
$a = "Hello world!";
echo "The value of variable 'a' before unset: " . $a . "<br>";
unset($a);
echo "The value of variable 'a' after unset: " . $a;
?>
```

Risultato:

The value of variable 'a' before unset: Hello world!

The value of variable 'a' after unset:

var_dump ()

La funzione **var_dump() scarica le informazioni su una o più variabili.**
Le informazioni contengono il tipo e il valore delle variabili.

```
var_dump ($ar);
/*
array(1) {
  ["secondo"]=>
  int(321)
}
 */
```

```
<?php
$a = 32;
echo var_dump($a) . "<br>";
$b = "Hello world!";
echo var_dump($b) . "<br>";
$c = 32.5;
echo var_dump($c) . "<br>";
$d = array("red", "green", "blue");
echo var_dump($d) . "<br>";
$e = array(32, "Hello world!", 32.5,
array("red", "green", "blue"));
echo var_dump($e) . "<br>";
// Dump two variables
echo var_dump($a, $b) . "<br>";
?>
```

Gestire le date in PHP

Succede molto di frequente di avere la necessità di lavorare con le date.

In PHP questo genere di operazioni sono davvero molto semplici grazie alla funzioni a disposizione.

in primis preoccuparsi di impostare il corretto timezone

```
date_default_timezone_set('Europe/Rome');
echo date_default_timezone_get();
```

Le principali funzioni per gestire le date

GESTIRE IL TIMESTAMP CON LA FUNZIONE TIME()

Il timestamp è un sistema di misurazione rappresentato da un intero che indica il numero di secondi a partire da una data definita Unix Epoch.

In particolare indica il numero di secondi intercorsi dal 1 gennaio 1970 alle ore 00:00 al momento corrente.

Per recuperare con PHP il timestamp attuale è sufficiente utilizzare la funzione time()

```
echo time(); // restituirà un valore simile a 1641313241
```

O da command

```
php -r "print time();"
```

La funzione strtotime()

OPERAZIONI CON LE DATE

non è possibile recuperare solo il timestamp corrente ma anche il timestamp di qualsiasi data.
In questo caso ci viene incontro la funzione strtotime().

Prende in ingresso una stringa che contiene il riferimento temporale che si vuole convertire:

```
echo strtotime('2021-01-01');
//1420070400
```

echo strtotime("now"); // timestamp corrente
echo strtotime("02 September 1978"); // timestamp del 02 settembre 1978
echo strtotime("+1 day"); // timestamp del giorno successivo a quello corrente
echo strtotime("+1 week"); // timestamp della settimana successiva al giorno corrente
echo strtotime("+1 week 2 days 4 hours 2 seconds"); // timestamp di una settimana, due giorni, 4 ore e 2 secondi rispetto al timestamp corrente
echo strtotime("next Thursday"); // timestamp del prossimo giovedì
echo strtotime("last Monday"); // timestamp dell'ultimo lunedì

Formattare una data

Un'altra operazione è quella di **formattare una specifica data in un formato da noi definito**.

Ad esempio, supponiamo di dover stampare un timestamp nel formato "gg/mm/aaaa h:i:s", per questo scopo possiamo utilizzare la funzione date()

La funzione date() prende in ingresso due parametri:

1. $format: il formato in cui vogliamo stampare la data;
2. $timestamp: il timestamp della data da stampare; se non viene passato questo valore verrà preso di default il timestamp corrente.

```
$data = strtotime("2021-12-05");
echo date("d/m/Y",$data);
```

Caratteri per formattare una data.

```php
<?php
print date("F j, Y, g:i a", strtotime("2022-01-05 09:25:12"));
//January 5, 2022, 9:25 am
```

j= giorno senza 0
i= minuto senza 0

tutti i formati:
https://www.php.net/manual/en/datetime.format.php

```php
print date("F j, Y, g:i a") . PHP_EOL;
print date("m.d.y") . PHP_EOL;
print date("j, n, Y") . PHP_EOL;
print date("Ymd") . PHP_EOL;
print date('h-i-s, j-m-y, it is w Day') . PHP_EOL;
print date('\i\t \i\s \t\h\e jS \d\a\y.') . PHP_EOL;
print date("D M j G:i:s T Y") . PHP_EOL;
print date('H:m:s \m \i\s\ \m\o\n\t\h') . PHP_EOL;
print date("H:i:s") . PHP_EOL;
print date("Y-m-d H:i:s") . PHP_EOL;

/*
February 6, 2022, 3:23 pm
02.06.22
6, 2, 2022
20220206
03-23-43, 6-02-22, 2328 2343 0 Sunpm22
it is the 6th day.
Sun Feb 6 15:23:43 UTC 2022
15:02:43 m is month
15:23:43
2022-02-06 15:23:43
*/
```

come convertire una data da un formato.

con la funzione date(), possiamo rappresentare una data nel formato di cui necessitiamo.

come convertire una data **da un formato, ad esempio quello anglosassone, nel nostro formato italiano**.

Il formato anglosassone è rappresentato come mese/giorno/anno ed è anche il formato standard riconosciuto dalla funzione strtotime()

```
$englishDate = '12/15/2016'; // 15 dicembre 2016 nel formato mm/dd/yyyy
$timestamp = strtotime($englishDate); // conterrà 1481756400
echo date('d/m/Y', $timestamp); // stamperà 15/12/2016
echo date('l, j F Y', $timestamp); // stamperà Thursday, 15 December 2016
```

date_create()

La funzione date_create() restituisce un nuovo oggetto DateTime.

```
<?php
$date=date_create("2021-03-15");
echo date_format($date,"Y/m/d");
?>
```

date_add()

La funzione date_add() **aggiunge alcuni giorni, mesi, anni, ore, minuti e secondi a una data.**
Valore di ritorno:
Restituisce un oggetto DateTime in caso di esito positivo.
FALSO in caso di fallimento

Nell'esempio aggiungiamo 40 giorni al 15 marzo

```
<?php
$date=date_create("2021-03-15");
date_add($date, date_interval_create_from_date_string("40 days"));
echo date_format($date,"Y-m-d");

//2021-04-24
```

date_diff()

La funzione date_diff() **restituisce la differenza tra due oggetti DateTime**.

```php
<?php
$date1=date_create("2021-03-15");
$date2=date_create("2021-12-12");
print_r (date_diff($date1,$date2));

/*
DateInterval Object
(
    [y] => 0
    [m] => 8
    [d] => 27
    [h] => 0
    [i] => 0
    [s] => 0
    [f] => 0
    [weekday] => 0
    [weekday_behavior] => 0
    [first_last_day_of] => 0
    [invert] => 0
    [days] => 272
    [special_type] => 0
    [special_amount] => 0
    [have_weekday_relative] => 0
    [have_special_relative] => 0
)
 */
```

date_format()

La funzione date_format() **restituisce una data formattata secondo il formato specificato.**
Nota: questa funzione non utilizza le impostazioni internazionali (tutto l'output è in inglese).

```
<?php
$date=date_create("2021-03-15");
echo date_format($date,"Y/m/d H:i:s");
?>
```

Il Risultato:

2021/03/15 00:00:00

date_parse_from_format()

La funzione date_parse_from_format() **restituisce un array associativo con informazioni dettagliate su una data specificata**, secondo il formato specificato.

```
<?php
print_r(date_parse_from_format("mdY","05122021"));
?>
```

Il Risultato:

```
Array
(
    [year] => 2021
    [month] => 5
    [day] => 12
    [hour] =>
    [minute] =>
    [second] =>
    [fraction] =>
    [warning_count] => 0
    [warnings] => Array
        (
        )

    [error_count] => 0
    [errors] => Array
        (
        )

    [is_localtime] =>
)
```

date_parse()

Restituisce un array associativo con informazioni dettagliate su una data specificata.

```
<?php
print_r(date_parse("2021-05-01 12:30:45.5"));
?>
```

Il Risultato:

```
Array
(
    [year] => 2021
    [month] => 5
    [day] => 1
    [hour] => 12
    [minute] => 30
    [second] => 45
    [fraction] => 0.5
    [warning_count] => 0
    [warnings] => Array
        (
        )

    [error_count] => 0
    [errors] => Array
        (
        )

    [is_localtime] =>
)
```

date_sub()

La funzione date_sub() **sottrae alcuni giorni, mesi, anni, ore, minuti e secondi da una data.**

```
<?php
$date=date_create("2021-03-15");
date_sub($date,date_interval_create_from_date_string("40 days"));
echo date_format($date,"Y-m-d");
?>
```

Il Risultato:

2021-02-03

getdate()

La funzione getdate() **restituisce informazioni su data/ora di un timestamp o la data/ora locale corrente.**

```
<?php
print_r(getdate());
?>
```

Il Risultato:

```
Array
(
    [seconds] => 37
    [minutes] => 35
    [hours] => 15
    [mday] => 6
    [wday] => 0
    [mon] => 2
    [year] => 2022
    [yday] => 36
    [weekday] => Sunday
    [month] => February
    [0] => 1644161737
)
```

mktime()

La funzione mktime() **restituisce il timestamp Unix per una data.**

```
mktime(
    int $hour,
    ?int $minute = null,
    ?int $second = null,
    ?int $month = null,
    ?int $day = null,
    ?int $year = null
): int|false
```

```
<?php
mktime(0,0,0,9,2,1978)
```

esempio con format:

```
print date("d/m/Y l", mktime(0,0,0,9,2,1978));
```

> 02/09/1978 Saturday

Funzioni con gli Array

count()

contare gli elementi

Supponiamo di avere un array di iscritti ad un evento e di volerne ottenere il numero, la funzione count() è esattamente quello di cui abbiamo bisogno.

```
$partecipanti = ['TuoNome', 'Gabriele', 'Renato', 'Giuseppe'];
echo "Ci saranno " . count($partecipanti) . " partecipanti all'evento";
```

L'esempio precedente restituirà:
Ci saranno 4 partecipanti all'evento

in_array()

PHP in_array : verifica il contenuto
supponiamo di voler verificare che l'utente TuoNome sia uno dei partecipanti, possiamo utilizzare la funzione in_array() che prende in ingresso due parametri:
il valore ricercato;
l'array in cui cercare.
ritorna un boolean true / false

```
in_array(mixed $needle, array $haystack, bool $strict = false): bool
```

```
$partecipanti = ['TuoNome', 'Gabriele', 'Renato', 'Giuseppe'];
if (in_array('TuoNome', $partecipanti)) {
echo "Simone è uno dei partecipanti";
} else {
echo «TuoNome non parteciperà all'evento";
}
```

Nel nostro caso l'output sarà: TuoNome è uno dei partecipanti.

shuffle()

Nel caso in cui avessimo la necessità di **mescolare gli elementi** all'interno di un array, abbiamo a disposizione la funzione **shuffle()**

```
$partecipanti = ['TuoNome', 'Gabriele', 'Renato', 'Giuseppe'];
shuffle($partecipanti);
print_r($partecipanti);
```

L'output che avremo sarà simile al seguente:

```
Array
(
[0] => Giuseppe
[1] => TuoNome
[2] => Renato
[3] => Gabriele
)
```

array_reverse()

PHP array_reverse : invertire l'ordine

```
$partecipanti = ['TuoNome', 'Gabriele',
'Renato', 'Giuseppe'];
$ordine_inverso =
array_reverse($partecipanti);
print_r($ordine_inverso);
```

Il risultato sarà:

```
Array
(
[0] => Giuseppe
[1] => Renato
[2] => Gabriele
[3] => TuoNome
)
```

array_merge()

Unire due array con il Merge
Dati due o più array, attraverso la funzione array_merge() possiamo avere un unico array che contiene gli elementi di ognuno di essi.
L'inserimento avviene in maniera da aggiungere gli elementi in coda all'array precedente. In caso di valori uguali l'ultimo valore andrà a sovrascrivere i precedenti.

```
$ar = array(1,2,5);
$ar2 = array_merge($ar, [6,8]);
print_r($ar2);
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 5
    [3] => 6
    [4] => 8
)
```

```
$partecipanti_lista_A = ['TuoNome', 'Gabriele'];
$partecipanti_lista_B = ['Renato', 'Giuseppe'];
$partecipanti_lista_C = ['Maria', 'Daniela'];
$partecipanti = array_merge($partecipanti_lista_A, $partecipanti_lista_B, $partecipanti_lista_C);
print_r($partecipanti);
```

La funzione array_merge() restituirà quindi il seguente risultato

```
Array
(
[0] => TuoNome
[1] => Gabriele
[2] => Renato
[3] => Giuseppe
[4] => Maria
[5] => Daniela
)
```

array_slice()

Estrarre una porzione di un array con Slice

Nel caso in cui volessimo **estrarre solo alcuni elementi da un array** possiamo usare la funzione array_slice()

prende in ingresso tre parametri:

- l'array;
- l'offset, ovvero l'elemento da cui iniziare l'estrazione;
- la lunghezza, ovvero il numero di elementi da estrarre a partire dall'offset.

```
$array = [1, 2, 3, 4, 5, 6, 7, 8, 9];
$output = array_slice($array, 0, 3);
print_r($output);
```

il risultato sarà:

```
Array
(
[0] => 1
[1] => 2
[2] => 3
)
```

array_chunk() dividere in blocchi di array

PHP array_chunk () Funzione

La funzione array_chunk() **divide un array in blocchi di nuovi array.**

```
<?php
$cars=array("Volvo","BMW","Toyota","Honda","Mercedes","O
pel");
print_r(array_chunk($cars,2));
```

```
Array
(
    [0] => Array
        (
            [0] => Volvo
            [1] => BMW
        )

    [1] => Array
        (
            [0] => Toyota
            [1] => Honda
        )

    [2] => Array
        (
            [0] => Mercedes
            [1] => Opel
        )

)
```

array_diff()

Funzione PHP array_diff()

La funzione array_diff() **confronta i valori di due (o più) array e restituisce le differenze.**

```
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");

$result=array_diff($a1,$a2);
print_r($result);
?>
```

Il Risultato:

Array ([d] => yellow)

array_fill ()

PHP array_fill () Funzione

La funzione array_fill() riempie gli elementi di un array con il valore specificato.
Valore di ritorno: Restituisce l'array pieno

array_fill(int $start_index, int $count, mixed $value): array

```
<?php
$a1=array_fill(3,4,"blue");
print_r($a1);
?>
```

Il Risultato:

Array ([3] => blue [4] => blue [5] => blue [6] => blue)
Array ([0] => red)

array_filter()

PHP array_filter () Funzione

La funzione array_filter() **filtra i valori di un array** utilizzando una funzione di callback o una closure.
Questa funzione passa ogni valore dell'array di input alla funzione di callback.
Se la funzione di callback restituisce true, il valore corrente dall'input viene restituito nell'array dei risultati.
Le chiavi dell'array vengono conservate.
Valore di ritorno: Restituisce l'array filtrato

& è un operatore bitwise
https://www.php.net/manual/en/language.operators.bitwise.php

guarda
http://easyonlineconverter.com/converters/bitwise-calculator.html

```
<?php
function test_odd($var)
 {
 return($var & 1);
 }
$a1=array(1,3,2,3,4);
print_r(array_filter($a1,"test_odd"));
?>
```

Il Risultato:

```
Array ( [0] => 1 [1] => 3 [3] => 3 )
```

```
$ar=[5,8,12,9,14,20,3,8,12,14,9,6,25];
print_r(array_filter($ar, function($v){
    return $v>10;
}));
```

```
Array
(
   [2] => 12
   [4] => 14
   [5] => 20
   [8] => 12
   [9] => 14
   [12] => 25
)
```

array_intersect ()

La funzione array_intersect() **confronta i valori di due (o più) array e restituisce le corrispondenze**.

Questa funzione confronta i valori di due o più array e restituisce un array che contiene le voci di array1 presenti in array2 , array3 e così via.

```
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");

$result=array_intersect($a1,$a2);
print_r($result);
?>
```

Il Risultato:

Array ([a] => red [b] => green [c] => blue)

array_key_exists()

La funzione array_key_exists() **controlla un array per una chiave specificata e restituisce true se la chiave esiste e false se la chiave non esiste.**
Suggerimento: ricorda che **se salti la chiave quando specifichi un array, viene generata una chiave intera**, che inizia da 0 e aumenta di 1 per ogni valore.

Nel esempio **controlliamo se la parola Volvo esiste** in un array.

```
<?php
$a=array("Volvo"=>"XC90","BMW"=>"X5");
if (array_key_exists("Volvo",$a))
  {
  echo "Key exists!";
  }
else
  {
  echo "Key does not exist!";
  }
?>
```

array_map()

La funzione array_map() invia ogni valore di un array a una funzione creata dall'utente e restituisce un array con nuovi valori, forniti dalla funzione creata dall'utente.

è possibile anche utilizzare una closure

```
$newAr = array_map(function($v){
        return $v *= 2;
},$ar);
```

```
<?php
function myfunction($v)
{
  return($v*$v);
}

$a=array(1,2,3,4,5);
print_r(array_map("myfunction",$a));
?>
Array
(
    [0] => 1
    [1] => 4
    [2] => 9
    [3] => 16
    [4] => 25
)
```

array_pop()

La funzione array_pop() **elimina e** ritorna l'ultimo elemento di un array.

```
$a=array("red","green","blue");
$eliminato = array_pop($a);
print_r($eliminato);
print_r($a);

blue
Array
(
    [0] => red
    [1] => green
)
```

array_push()

La funzione array_push() inserisce uno o più elementi alla fine di un array.

ritorna il numero (count) degli elementi dell'array

```php
<?php
$a=array("red","green");
array_push($a,"blue","yellow");
print_r($a);
```

```
Array
(
    [0] => red
    [1] => green
    [2] => blue
    [3] => yellow
)
```

array_replace()

La funzione array_replace() **sostituisce i valori del primo array con i valori degli array successivi**.
ritorna l'array con i valori sostituiti

```php
<?php
$ar1 = array("gatto", "cane",
"cavallo", "elefante");
$ar2 = array("topo","scimmia");
print_r(
    array_replace($ar1,$ar2)
);
/*
Array
(
    [0] => topo
    [1] => scimmia
    [2] => cavallo
    [3] => elefante
)*/
```

array_replace() - array associativi

Funzione PHP array_replace()
Se una chiave di array1 esiste in array2, i valori di array1 verranno sostituiti dai valori di array2.

Se la chiave esiste solo in array1, verrà lasciata così com'è .

```
<?php
$a1=array("a"=>"red","b"=>"green");
$a2=array("a"=>"orange","burgundy");
print_r(array_replace($a1,$a2));
?>
```

Il Risultato:
Array ([a] => orange [b] => green [0] => burgundy)

array_search ()

La funzione array_search() cerca un valore nei valori di un array e restituisce la chiave

```php
<?php
$ar = array("primo"=>"rosso",
"secondo"=>"verde",
"terzo"=>"giallo");
echo array_search("rosso",$ar);
//primo

echo
array_search("marrone",$ar);
//null
?>
```

array_shift()

La funzione array_shift() rimuove il primo elemento da un array e restituisce il valore dell'elemento rimosso.

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_shift($a);
print_r ($a);
?>
```

Il Risultato:

```
red //valore restituito
Array ( [b] => green [c] => blue )
```

array_unique()

La funzione array_unique() **rimuove i valori duplicati** da un array. Se due o più valori dell'array sono uguali, **il primo verrà mantenuto e l'altro verrà rimosso**.

Nota: l'array restituito manterrà il tipo di chiave del primo elemento dell'array.

Valore di ritorno: Restituisce l'array filtrato.

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"red");
print_r(array_unique($a));
?>
```

IL Risultato

Array ([a] => red [b] => green)

arsort()

La funzione arsort() **ordina un array associativo in ordine decrescente**, in base al valore.

Suggerimento: utilizzare la funzione **asort**() per ordinare un array associativo in **ordine crescente**, in base al valore.

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
arsort($age);
foreach($age as $x=>$x_value)
  {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
  }
?>
```

Il Risultato:

Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35

asort()

La funzione arsort() **ordina un array associativo in ordine** crescente, in base al valore.

```
<?php
    $ar=array("Ben"=>"37",
"Joe"=>"43", "Peter"=>"35");
    asort($ar);
    print_r($ar);
?>
Array
(
    [Peter] => 35
    [Ben] => 37
    [Joe] => 43
)
```

list()

La funzione list() viene utilizzata per assegnare valori a un elenco di variabili in un'operazione.

```php
<?php
$info = array('coffee', 'brown', 'caffeine');

// Listing all the variables
list($drink, $color, $power) = $info;
echo "$drink is $color and $power makes it sp
ecial.\n";
```

array_keys() / array_values()

Dato **un array multidimensionale** con coppie chiave/valori abbiamo a disposizione due funzioni, **array_keys()** e **array_values()**, che ci permettono rispettivamente di **restituire un array di chiavi e di valori** dell'array.

```
$array = [
'TuoNome' => 29,
'Marco' => 28,
'Michele' => 35,
'Luca' => 22
];
$nomi = array_keys($array);
$anni = array_values($array);
print_r($nomi);
print_r($anni);
```

L'output dei due array sarà:

```
Array
(
[0] => TuoNome
[1] => Marco
[2] => Michele
[3] => Luca
)
Array
(
[0] => 29
[1] => 28
[2] => 35
[3] => 22
)
```

FILTER_VAR() - Validazione dei dati in PHP

strumenti di PHP **per validare e/o "ripulire" dati** attraverso **la funzione filter_var()**
Esistono due macrotipi di filtri che possiamo utilizzare con essa:

1. **validation**: verificano che un dato sia o meno valido;
2. **sanitizazion**: modificano il dato per renderlo valido.

VALIDATION DEI DATI

La validazione di un indirizzo email è probabilmente il caso più comune di utilizzo di filter_var():

```
$emailValida = "test@gmail.com";
$emailNonValida = "invalid.email";
if (filter_var($emailValida,
FILTER_VALIDATE_EMAIL)) {
    echo "$emailValida valida";
}
if (filter_var($emailNonValida,
FILTER_VALIDATE_EMAIL)) {
    echo "$emailNonValida valida";
}
```

FILTER_VALIDATE

```
if (filter_var('aafasdfsdfa@bbbb.it', FILTER_VALIDATE_EMAIL)) {
    echo 'VALID';
} else {
    echo 'NOT VALID';
}
```

Filtro	Descrizione
FILTER_VALIDATE_BOOLEAN	Restituisce true per i valori "1", "true", "on" e "yes".
FILTER_VALIDATE_EMAIL	Verifica la validità di una mail.
FILTER_VALIDATE_FLOAT	Verifica che una variabile sia un float valido.
FILTER_VALIDATE_INT	Verifica che la variabile sia un intero valido. È possibile anche verificare che sia compreso in un range.
FILTER_VALIDATE_IP	Verifica la validità di un IP.
FILTER_VALIDATE_REGEXP	Verifica che la variabile sia valida per un'espressione regolare.
FILTER_VALIDATE_URL	Verifica la validità di un URL

https://www.php.net/manual/en/filter.filters.validate.php

FILTER_VALIDATE_INT

Abbiamo detto che in FILTER_VALIDATE_INT è possibile definire un range di interi in cui la variabile dovrebbe essere compresa.
Vediamo qualche esempio di utilizzo creando un array di numeri e passandoli ad una funzione che ne verifica la validità.
Supponiamo inoltre che un valore sia valido se intero e compreso tra 1 e 5.

```php
$valori = [
    '1',
    '-1',
    '2.0',
    'asd',
    '0',
    '5',
    '10',
];
function validaIntero($valore) {
    $options = [
        'options' => [
            'min_range' => 1,
           'max_range' => 5,
        ]
    ];
    if (filter_var($valore, FILTER_VALIDATE_INT, $options)) {
        echo "Il valore $valore è valido\n";
    } else {
        echo "Il valore $valore non è valido\n";
    }
}
foreach($valori as $valore) {
    validaIntero($valore);
}
```

Il risultato che otteremo:

Il valore 1 è valido
Il valore -1 non è valido
Il valore 2.0 non è valido
Il valore asd non è valido
Il valore 0 non è valido
Il valore 5 è valido
Il valore 10 non è valido

Sanitization dei dati

Oltre che validare un dato possiamo anche modificarlo affinché il PHP provi a renderlo valido:

```php
$email1 = "test@gmail.com";
$email2 = "(test@gmail.com)";
$email3 = "testATgmail.com";
function sanitizzaEmail($email) {
    $emailSanitizzata = filter_var($email, FILTER_SANITIZE_EMAIL);
    $emailValida = filter_var($emailSanitizzata, FILTER_VALIDATE_EMAIL);
    echo "Indirizzo email: $email\n";
    echo "Indirizzo email sanitizzato: $emailSanitizzata\n";
    echo "Email valida: " . ($emailValida ? 'SI' : 'NO') . "\n\n";
}
sanitizzaEmail($email1);
sanitizzaEmail($email2);
sanitizzaEmail($email3);
```

Il Risultato:

Indirizzo email: test@gmail.com
Indirizzo email sanitizzato: test@gmail.com
Email valida: SI
Indirizzo email: (test@gmail.com)
Indirizzo email sanitizzato: test@gmail.com
Email valida: SI
Indirizzo email: testATgmail.com
Indirizzo email sanitizzato: testATgmail.com
Email valida: NO

Nel primo caso avevamo una mail valida quindi era facile aspettarsi un risultato positivo. Nel secondo PHP ha provato a modificare il valore e ha restituito un'email valida. L'ultimo caso non ha avuto risultato positivo neppure dopo aver applicato FILTER_SANITIZE_EMAIL

```php
$email = filter_var('aafasdfsdfa@bbb"b',
FILTER_SANITIZE_EMAIL);
echo 'Mail: ' . $email; //aafasdfsdfa@bbbb
```

FILTER_SANITIZE

Come per la validazione, anche la sanitizzazione dei dati ha diversi filtri.

Filtro	Descrizione
`FILTER_SANITIZE_EMAIL`	Rimuove tutti i caratteri eccetto lettere, numeri e !#$%&'*+-/=?^_`{
`FILTER_SANITIZE_ENCODED`	Codifica un URL.
`FILTER_SANITIZE_MAGIC_QUOTES`	Applica la funzione `addslashes()` alla stringa.
`FILTER_SANITIZE_NUMBER_FLOAT`	Rimuove tutti i caratteri eccetto numeri e i simboli + e -
`FILTER_SANITIZE_NUMBER_INT`	Rimuove tutti i caretteri eccetto i simboli + e -
`FILTER_SANITIZE_SPECIAL_CHARS`	Escape HTML dei caratteri '"<>&
`FILTER_SANITIZE_FULL_SPECIAL_CHARS`	Equivale a richiamare la funzione `htmlspecialchars()` con il parametro `ENT_QUOTES` .
`FILTER_SANITIZE_STRING`	Rimuove tutti i tags, opzionalmente codifica i caratteri speciali.
`FILTER_SANITIZE_STRIPPED`	Alias del filtro `FILTER_SANITIZE_STRING` .
`FILTER_SANITIZE_URL`	Rimuove tutti i caratteri eccetto lettere, numeri e $-_.+!*'(),{}
`FILTER_UNSAFE_RAW`	Non compie operazioni, opzionalmente rimuove o codifica caratteri speciali.

sanitize e injection

CON filter_var abbiamo prevenuto questo attacco., in ogni caso il comportamento migliore in questo caso sarebbe utilizzare mysql_real_escape_string

```php
<?php
//se l'utente inserisse nei dati:
$user = "' or '1' = '1";
$pass = "' or '1' = '1";
$user = filter_var($user,
FILTER_SANITIZE_ADD_SLASHES);
$pass = filter_var($pass,
FILTER_SANITIZE_ADD_SLASHES);
$sql ="SELECT * FROM users WHERE name='$user' AND
password='$pass";

//la query sarebbe

print $sql;
//SELECT * FROM users WHERE name='' or '1' = '1' AND
password='' or '1' = '1'
//ovvero sempre vera e estrarrebbe tutti i record
```

```php
<?php
//se l'utente inserisse nei dati:
$user = "' or '1' = '1";
$pass = "' or '1' = '1'";

$sql ="SELECT * FROM utenti WHERE name='$user'
AND password='$pass";

//la query sarebbe

print $sql;
//SELECT * FROM utenti WHERE name='' or '1' =
'1' AND password='' or '1' = '1'
//ovvero sempre vera e estrarrebbe tutti i
record
```

FILTER_SANITIZE_STRING

Il filtro **FILTER_SANITIZE_STRING** rimuove i tag e **rimuove o codifica i caratteri speciali da una stringa.**

Possibili opzioni e flag:

FILTER_FLAG_NO_ENCODE_QUOTES - Non codificare le virgolette

FILTER_FLAG_STRIP_LOW - Rimuove i caratteri con valore ASCII < 32

FILTER_FLAG_STRIP_HIGH - Rimuove i caratteri con valore ASCII > 127

FILTER_FLAG_ENCODE_LOW - Codifica caratteri con valore ASCII < 32

FILTER_FLAG_ENCODE_HIGH - Codifica caratteri con valore ASCII > 127

FILTER_FLAG_ENCODE_AMP - Codifica il carattere "&" in &

```
<?php
$str = "<h1>Hello World!</h1>";

$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Risultato: Hello World!

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

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

FILTER_SANITIZE_STRING DEPRECATO DALLA VERSIONE 8 USARE htmlspecialchars

```php
$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";
}
```

UPLOAD FILES – move_uploaded_file

move_uploaded_file è utilizzato per caricare un file ricevuto dall'array FILES

```
<form action="indexa.php" method="POST" enctype="multipart/form-data">
        <div class="mb-3">
            <label for="formFile" class="form-label">Default file
input example</label>
            <input class="form-control" type="file" id="formFile"
name="formFile">
        </div>
        <input type="submit" value="invia">
    </form>
```

```
    if (isset ($_FILES['formFile']) ){
        $uploaded =
move_uploaded_file($_FILES['formFile']['tmp_name'],
__DIR__ . "/" . $_FILES['formFile']['name']);
        if($uploaded) {
            print "file uploaded";
        }
    }
```

esempio exploit xss

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that any JavaScript code can be added inside the <script> tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?
$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Gestire i cookie con PHP

I cookie sono una sorta di identificativo che viene utilizzato dai siti Web per memorizzare informazioni relative agli utenti.
Questo strumento ci consente, ad esempio, di riconoscere se un utente è ancora loggato sul sito oppure no. Altri comportamenti utili dei cookie possono memorizzare alcune azioni come la richiesta di chiudere un banner pubblicitario così da non visualizzarlo alla prossima visita.
I cookie vengono memorizzati automaticamente dal browser, quindi fino alla cancellazione continueranno ad identificare l'utente. È possibile anche impostare una data di scadenza in modo che sia il sito Web a decidere per quanto tempo salvare questa informazione. Si può decidere infatti di tenere memorizzato un cookie fino alla chiusura del browser.

In PHP i cookie sono memorizzati all'interno dell'array riservato $_COOKIE. Effettuando quindi un dump dell'array otterremo un'informazione simile alla seguente:

```
var_dump($_COOKIE);
array(3) {
   ["__utma"]=>
   string(55)
"111872281.2010784770.1481539931.1481977233.1481996919.4"
   ["__utmc"]=>
   string(9) "111872281"
   ["__utmz"]=>
   string(70)
"111872281.1481539931.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)"
}
```

Ovviamente questi valori differiscono in base alla pagina che stiamo visitando, all'utente che accede e a diversi altri fattori.
Nel nostro esempio ci sono tre cookie con nome __utm* (che corrisponde alla chiave dell'array) e relativi valori.

$_COOKIE [] Accedere al valore di un cookie

Per accedere al valore di un singolo cookie non abbiamo bisogno d'introdurre nuove funzioni ma possiamo accedere al singolo elemento dell'array.
Supponiamo di avere un cookie chiamato user_id, possiamo stampare il suo valore con:

```
echo $_COOKIE['user_id'];
```

setcookie() Aggiungere un nuovo cookie

Per generare un nuovo cookie dobbiamo introdurre la funzione **setcookie()** con la quale impostare anche la durata oltre al nome e al valore.

Prima di proporre un esempio, però, è bene specificare che i cookie fanno parte dell'header di una risposta HTTP, quindi è necessario eseguire queste operazioni prima che venga inviata la risposta del server.

In termini più semplici bisognerebbe **compiere le operazioni di scrittura dei cookie prima che venga prodotto qualsiasi output**.

Supponiamo di voler salvare l'id di un utente all'interno di un cookie:

```
setcookie("user_id", "345", strtotime("+1 year"));
```

La funzione setcookie()

La funzione **setcookie()** prende in ingresso i parametri:

Parametro	Descrizione
name	Nome del cookie, nel nostro caso `user_id` .
value	Valore del cookie (`345`).
expire	Scadenza del cookie (1 anno).
path	Percorso per cui è valido. Di default è impostato su tutto il sito, ma se ad esempio scegliamo `/sottodirectory/` esso sarà valido solo per quel percorso.
domain	Dominio per cui è valido.
secure	Indica se il cookie dovrebbe essere trasmesso attraverso una connessione HTTPS.
httponly	Indica se il cookie è accessibile solo attraverso il protocollo HTTP. Questo implica, ad esempio, che se il valore è `true` il cookie non è accessibile da Javascript.

Modificare o Eliminare un Cookie

PER MODIFICARE qualsiasi informazione relativa ad un cookie è sufficiente **richiamare la funzione setcookie()** con i nuovi dati.

Ad esempio per cambiare lo user id possiamo richiamarlo come segue:

```
setcookie("user_id", "123", strtotime("+1 year"));
```

ELIMINARE UN COOKIE

Per eliminare un cookie impostare una data negativa

```
setcookie("user_id", "123", time()-1);
```

Gestire le sessioni in PHP

Le sessioni sono un meccanismo alternativo ai cookie per memorizzare informazioni relative all'utente.

A differenza dei cookie che vengono salvati sul client, ovvero sul dispositivo dell'utente tramite browser, la sessione memorizza le informazioni sul server che ospita l'applicazione.
Di default la sessione salva i dati in file testuali salvati nel file system del server. È possibile, però, modificare tale configurazione in maniera da salvarle anche su database, Memcache, Redis, e così via.
Tutte alternative molto utili quando si lavora, ad esempio, con applicazioni che risiedono in server differenti.
L'associazione utente/sessione avviene nella maggior parte dei casi attraverso un cookie che contiene l'id della sessione.
La durata di una sessione, così come avviene per i cookie, può essere decisa dallo sviluppatore. Se non diversamente specificato essa scadrà alla chiusura del browser.
L'uso più comune che viene fatto di una sessione è quello di implementare un sistema di autenticazione su un sito Web. Una volta verificato che le credenziali di accesso sono valide, infatti, è utile memorizzare le informazioni dell'utente su una sessione così da non doverle ricaricare da database all'apertura di ogni pagina.

Iniziare una sessione

Una sessione, per poter essere utilizzata, ha bisogno di essere inizializzata.

Come per i cookie è necessario inizializzarla prima che qualsiasi output sia già stato inviato alla pagina.

La funzione per inizializzare una sessione si chiama session_start().

Essa prende in ingresso un array di opzioni che, se impostate, vanno a sovrascrivere quelle di default impostate nel php.ini.
Per sapere quali sono tali impostazioni si può fare riferimento alla documentazione ufficiale.
Una volta inizializzata la sessione possiamo accedere ai dati relativi e, di conseguenza, poterne aggiungere altri.

```php
<?php session_start(); //sessione inizializzata
```

Accedere ai dati di una sessione

Una volta inizializzata una sessione, le informazioni sono salvate all'interno dell'array $_SESSION

Supponiamo quindi di voler MEMORIZZARE IL NOME DELL'UTENTE IN SESSIONE, possiamo creare una chiave nell'array:

```php
<?php session_start(); $_SESSION['name'] = 'TuoNome'; echo $_SESSION['name'];
//stamperà il nome in sessione
```

Come per i cookie, è possibile RIMUOVERE **UNA PROPRIETÀ** DALLA SESSIONE attraverso l'unset su di essa:

```php
<?php session_start(); $_SESSION['name'] = 'TuoNome';
unset($_SESSION['name']);
```

Se volessimo RIMUOVERE **TUTTE LE VARIABILI** IN SESSIONE, invece, possiamo utilizzare la funzione apposita:

```php
<?php session_start(); $_SESSION['name'] = 'TuoNome'; session_unset();
```

Distruggere una sessione;
Mantenere una sessione tra diversi file.

Una volta che **la sessione non è più necessaria**, ad esempio al logout di un utente, possiamo **distruggerla** attraverso la funzione session_destroy():

```php
<?php session_start(); $_SESSION['name'] = 'TuoNome'; session_destroy(); //a questo punto la sessione non è più disponibile per l'utente
```

MANTENERE UNA SESSIONE TRA DIVERSI FILE

Una sessione inizializzata in un file, ad esempio login.php, **non viene mantenuta automaticamente se accediamo ad un'altra pagina.**
Ragione per cui **all'inizio di ogni file che utilizziamo è necessario richiamare session_start()** che, nel caso in cui la sessione sia già stata creata, la recupera con i dati salvati in precedenza.

Database PHP MySQL

Con PHP puoi connetterti e manipolare i database.
MySQL è il sistema di database più popolare utilizzato con PHP.

Cos'è MySQL?

MySQL è **un sistema di database utilizzato sul web**
MySQL è un sistema di database che **gira su un server**
MySQL è ideale sia **per applicazioni piccole che grandi**
MySQL è molto **veloce, affidabile e facile da usare**
MySQL utilizza SQL standard
MySQL si compila su una serie di piattaforme
MySQL può essere scaricato e utilizzato gratuitamente
MySQL è sviluppato, distribuito e supportato da Oracle Corporation
MySQL prende il nome dalla figlia del co-fondatore Monty Widenius: My

mysql - connessione da riga comando

C:\xampp\mysql\bin>mysql -u root -p -h localhost nomedb
Enter password:
>show tables;
il ; permette di eseguire l'istruzione che altrmenti non sarebbe terminata

oppure
C:\xampp\mysql\bin>mysql -u root -p -h localhost
Enter password:
> use nomedb
>show tables;

se da cmd mysql non è definito aggiungere nelle variabili di ambiente ad es. x windows:

Right click “My Computer”
Click “Properties”
Select “Advanced”
Click “Environment Variables”
Locate “System variables”
Select the “Path” variable
Click “Edit”
…”;C:\Program Files\MySQL\MySQL Server 5.6\bin”

mysqli_connect ()

La funzione connect() / mysqli_connect() apre una nuova connessione al server MySQL.

Sintassi
Stile orientato agli oggetti:

```
$mysqli = new mysqli($host, $username, $password, $dbname);
if ($mysqli->connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli->connect_error;
  die;
}
```

Stile procedurale:

```
$conn = mysqli_connect($host, $username, $password, $dbname);
if ( mysqli_connect_errno( $conn ) ) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
```

Valore di ritorno:
Restituisce un oggetto che rappresenta la connessione al server MySQL

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

// Check connection
if ($mysqli-> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}
?>
```

mysqli_connect_errno()

La funzione connect_errno / mysqli_connect_errno() restituisce il codice di errore dell'ultimo errore di connessione, se presente.
Valore di ritorno:
Restituisce un valore del codice di errore.
Zero se non si è verificato alcun errore

Object-oriented style
int *$mysqli->errno*;
Procedural style
mysqli_errno(mysqli $mysql):
int

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "wor
ld");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_errorno);
    exit();
}

$link = mysqli_connect("localhost", "my_user", "my_password", "w
orld");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_errno());
    exit();
}
```

mysqli_connect_error()

La funzione connect_error / mysqli_connect_error() **restituisce la descrizione dell'errore dall'ultimo errore di connessione, se presente**.
Valore di ritorno:
Restituisce una stringa che descrive l'errore.
NULL se non si è verificato alcun errore

Object-oriented style
?string *$mysqli->connect_error*;

Procedural style
```
mysqli_connect_error(): ?string
```

```php
<?php
/* @ is used to suppress default error messages */
$mysqli = @new mysqli('localhost', 'fake_user', 'my_passwo
rd', 'my_db');

if ($mysqli->connect_error) {
    /* Use your preferred error logging method here */
    error_log('Connection error: ' . $mysqli-
>connect_error);
}

/* @ is used to suppress default error messages */
$link = @mysqli_connect('localhost', 'fake_user', 'my_password',
 'my_db');

if (!$link) {
    /* Use your preferred error logging method here */
    error_log('Connection error: ' . mysqli_connect_error());
}
```

mysqli_query()

La funzione query() / mysqli_query() esegue una query su un database.

Object-oriented style

```
public mysqli::query(string $query, int $result_mode = MYSQLI_STORE_RESULT): mysqli_result|bool
```

Procedural style

```
mysqli_query(mysqli $mysql, string $query, int $result_mode = MYSQLI_STORE_RESULT): mysqli_result|bool
```

```php
<?php
/* Select queries return a resultset */
$result = $mysqli->query("SELECT Name FROM City LIMIT 10");

/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
$result = mysqli_query($link, "SELECT * FROM City", MYSQLI_USE_RESULT);
```

mysqli_affected_rows()

La funzione mysqli_affected_rows restituisce il numero di righe interessate nella precedente query SELECT, INSERT, UPDATE, REPLACE o DELETE.

Object-oriented style

```
int|string $mysqli->affected_rows;
```

Procedural style

```
mysqli_affected_rows(mysqli $mysql): int|string
```

```php
<?php
/* update rows */
$mysqli->query("UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", $mysqli->affected_rows);
```

```php
/* select all rows */
$result = mysqli_query($link, "SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", mysqli_affected_rows($link));
```

mysqli_close()

La funzione close() / mysqli_close() **chiude una connessione al database aperta in precedenza**.

Object-oriented style

```
public mysqli::close(): bool
```

Procedural style

```
mysqli_close(mysqli $mysql): bool
```

```php
<?php
 /* Close the connection as soon as
  it's no longer needed */
$mysqli->close();
```

```php
 /* Close the connection as soon as
  it's no longer needed */
mysqli_close($mysqli);
```

mysqli_fetch_array ()

La funzione fetch_array() / **mysqli_fetch_array()** **recupera una riga di risultato come un array associativo**, un array numerico o entrambi.
Nota: i nomi dei campi restituiti da questa funzione fanno distinzione tra maiuscole e minuscole.
DEPRECATA DA PHP 5.5

```
mysql_fetch_array(resource $result, int $result_type = MYSQL_BOTH): array
```

```
<?php
$sql = "SELECT Lastname, Age FROM Persons
ORDER BY Lastname";
$result = $mysqli -> query($sql);
// Numeric array
$row = $result -> fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
// Associative array
$row = $result -> fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Lastname"],
$row["Age"]);
// Free result set
$result -> free_result();
$mysqli -> close();
?>
```

mysqli_fetch_assoc ()

La funzione fetch_assoc() / mysqli_fetch_assoc() recupera una riga di risultati come array associativo.

Nota: i nomi dei campi restituiti da questa funzione fanno distinzione tra maiuscole e minuscole.

Object-oriented style

```
public mysqli_result::fetch_assoc(): array|null|false
```

Procedural style

```
mysqli_fetch_assoc(mysqli_result $result): array|null|false
```

<?php

Object-oriented style

```
$query = "SELECT Name, CountryCode FROM City ORDER BY ID DESC";

$result = $mysqli->query($query);

/* fetch associative array */
while ($row = $result->fetch_assoc()) {
    printf("%s (%s)\n", $row["Name"], $row["CountryCode"])
;
}
```

Procedural style

```
$query = "SELECT Name, CountryCode FROM City ORDER BY ID DESC";

$result = mysqli_query($mysqli, $query);

/* fetch associative array */
while ($row = mysqli_fetch_assoc($result)) {
    printf("%s (%s)\n", $row["Name"], $row["CountryCode"]);
}
```

mysqli_insert_id()

La funzione mysqli_insert_id() restituisce l'id (generato con AUTO_INCREMENT) dall'ultima query.
Valore di ritorno:

Un numero intero che rappresenta il valore del campo AUTO_INCREMENT aggiornato dall'ultima query.

Restituisce zero se non ci sono stati aggiornamenti o nessun campo AUTO_INCREMENT

Object-oriented style

```
int|string $mysqli->insert_id;
```

Procedural style

```
mysqli_insert_id(mysqli $mysql): int|string
```

```php
<?php
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
$mysqli->query($query);

printf("New record has ID %d.\n", $mysqli->insert_id);
```

```php
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
mysqli_query($link, $query);

printf("New record has ID %d.\n", mysqli_insert_id($link));
```

mysqli_prepare()

La funzione prepare() / mysqli_prepare() viene utilizzata per preparare un'istruzione SQL per l'esecuzione.

La separazione fra la preparazione dell'SQL e i dati ci permette di avere una protezione rispetto alle SQL injection, infatti i dati provenienti dall'utente verranno gestiti al di fuori dell'istruzione SQL

stile procedurale

```
$stmt = mysqli_stmt_init($conn);
mysqli_stmt_prepare($stmt, "INSERT INTO utenti (nome,cognome) VALUES (?,?)");
mysqli_stmt_bind_param($stmt, "ss", $nome, $cognome );
mysqli_stmt_execute($stmt);
mysqli_stmt_close($stmt);
```

stile oggetti:

```
$stmt = $mysqli->stmt_init();
$stmt = $mysqli->prepare("INSERT INTO utenti (nome,cognome) VALUES (?,?)");
$stmt->bind_param("ss", $nome, $cognome);
$stmt->execute();
$stmt->close();
```

types(Mandatory)
A string (of individual characters) specifying the types of the variables where −

- **i** represents an integer type
- **d** represents an double type
- **s** represents an string type
- **b** represents an blob type

mysqli_real_escape_string()

La funzione real_escape_string() / **mysqli_real_escape_string() esegue l'escape dei caratteri speciali in una stringa da utilizzare in una query SQL**, tenendo conto del set di caratteri corrente della connessione.
Questa funzione viene utilizzata per creare una stringa SQL legale che può essere utilizzata in un'istruzione SQL.

stile procedurale:

```
$nome = mysqli_real_escape_string($conn, $_GET["nome"] ?? null);
$cognome = mysqli_real_escape_string($conn, $_GET["cognome"] ?? null);
$sql = "INSERT INTO utenti (nome,cognome) VALUES ('$nome','cognome')";
print $sql;
if (!mysqli_query($conn, $sql)) {
  printf("%d Righe inserite:", mysqli_affected_rows($conn));
}
```

stile oggetti:

```
$nome = $mysqli->real_escape_string($_GET["nome"] ?? null);
$cognome = $mysqli->real_escape_string($_GET["cognome"] ?? null);
$sql = "INSERT INTO utenti (nome,cognome) VALUES ('$nome','cognome')";
if (!$mysqli->query($sql)) {
  printf("%d Righe inserite:", $mysqli->affected_rows);
}
```

mysqli_autocommit () mysqli_commit() mysqli_rollback ()

La funzione rollback() / mysqli_rollback() esegue il rollback della transazione corrente per la connessione al database specificata.
-Specifica la connessione Mysql da usare.
-Valore di ritorno:
VERO sul successo.
FALSO in caso di fallimento

procedural style:

```
mysqli_autocommit($conn, false);
....
mysqli_commit($conn);
mysqli_commit($conn);
```

object style:

```
$mysqli->autocommit(false);
.....
$mysqli->rollback();
$mysqli->commit();
```

mysqli_commit()

La funzione commit() / mysqli_commit() esegue il commit della transazione corrente per la connessione al database specificata.

Object-oriented style

```
public mysqli::commit(int $flags = 0, ?string $name = null): bool
```

Procedural style

```
mysqli_commit(mysqli $mysql, int $flags = 0, ?string $name = null): bool
```

```
<?php
$mysqli->autocommit(FALSE);

try{

  $mysqli->query("INSERT INTO myCity (id) VALUES (100)")
or throw new Exception('error!');

// or we can use

  if( !$mysqli->query("INSERT INTO myCity (id) VALUES
(200)"){
    throw new Exception('error!');
  }

}catch( Exception $e ){
  $mysqli->rollback();
}
$mysqli->commit();

OPPURE PROCEDURAL STYLE
$con = mysqli_connect("host", "username", "password", "database") or
die("Could not establish connection to database");
$mysqli->autocommit(false);
…
$query = mysqli_query($con, "INSERT INTO users (username) VALUES
('$user') ");
…..
//Make a one-time hit to the database
mysqli_commit($con)
```

mysqli_select_db ()

La funzione select_db() / **mysqli_select_db()** viene **utilizzata per modificare il database predefinito per la connessione.**

Parametri:

Specifica la connessione Mysql da usare.

Specifica il nome del database.

Valore di ritorno:

VERO sul successo.

FALSO in caso di fallimento

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");
if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}

// Return name of current default database
if ($result = $mysqli -> query("SELECT DATABASE()")) {
  $row = $result -> fetch_row();
  echo "Default database is " . $row[0];
  $result -> close();
}
// Change db to "test" db
$mysqli -> select_db("test");
// Return name of current default database
if ($result = $mysqli -> query("SELECT DATABASE()")) {
  $row = $result -> fetch_row();
  echo "Default database is " . $row[0];
  $result -> close();
}
$mysqli -> close();
?>
```

mysqli_set_charset()

La funzione set_charset() / **mysqli_set_charset() specifica il set di caratteri predefinito** da utilizzare quando si inviano dati da e verso il server del database.
Nota: affinché questa funzione funzioni su una piattaforma Windows, è necessaria la libreria client MySQL 4.1.11 o successiva (per MySQL 5.0, è necessaria la 5.0.6 o successiva).
Parametri:
Specifica la connessione Mysql da usare.
Specifica il set di caratteri predefinito.

```
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");
if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}
echo "Initial character set is: " . $mysqli -> character_set_name();
// Change character set to utf8
$mysqli -> set_charset("utf8");
echo "Current character set is: " . $mysqli -> character_set_name();
$mysqli -> close();
?>
```

PDO::__construct

DSN

Il nome dell'origine dati, o DSN, **contiene le informazioni necessarie per la connessione al database**.
In generale, un DSN è costituito dal nome del driver PDO, seguito da due punti, seguito dalla sintassi di connessione specifica del driver PDO.

Opzioni

Una matrice chiave=>valore di opzioni di connessione specifiche del driver.
PDO::__construct() genera una PDOException se il tentativo di connessione al database richiesto non riesce, indipendentemente da quale PDO::ATTR_ERRMODE è attualmente impostato.

potrebbe essere anche una uri:

```
$dsn = 'uri:file:///usr/local/dbconnect';
```

```
$dsn =
'mysql:host=localhost;dbname=test;charset=UTF8';
    $user = 'root';
    $password = '';

    try{
        $pdo = new PDO($dsn, $user, $password);
    }catch (PDOException $e){
        die (message('danger', $e->getMessage()));
    }
```

PDO:: avvia una transazione

Descrizione

```
public PDO::beginTransaction (): bool
```

Disattiva la modalità di commit automatico.
Mentre la modalità autocommit è disattivata, **le modifiche apportate al database tramite l'istanza dell'oggetto PDO non vengono salvate finché non si termina la transazione chiamando PDO::commit()** .

La chiamata a PDO::**rollBack()** eseguirà il rollback di tutte le modifiche al database e riporterà la connessione in modalità autocommit.

Valori di ritorno: Restituisce true in caso di successo o false in caso di fallimento.
Errori/Eccezioni: Genera un'eccezione PDOException se è già stata avviata una transazione o se il driver non supporta le transazioni.

```
$pdo->beginTransaction();
```

PDO::commit()

effettua una commit

Nota : non tutti i database consentiranno alle transazioni di operare su istruzioni DDL: alcuni genereranno errori, mentre altri (incluso MySQL) eseguiranno automaticamente il commit della transazione dopo che è stata rilevata la prima istruzione DDL.

```php
<?php
      try {
        $pdo->beginTransaction();
        $prepared = $pdo->prepare("DELETE FROM servizi WHERE id_servizio = :id_servizio limit 1");
        $prepared->bindParam(':id_servizio', $_REQUEST["id_servizio"]);
        $prepared->execute();
        $pdo->commit();
      }catch (PDOException $e){
        $pdo->rollBack();
        die (message('danger', $e->getMessage()));
      }
```

PDO::query()

PDO::query — Prepara ed esegue un'istruzione SQL senza segnaposto

```
public PDO::query(string $query, ?int $fetchMode = null): PDOStatement|false
```

```
try{
        $stm = $pdo->query('SELECT * FROM servizi');
        $rows = $stm->fetchAll();
    }catch (Exception $e){
        die (message('danger', $e->getMessage()));
    }
    // iterate over array by index and by name
    foreach($rows as $row) {
```

PDO::errorCode() e errorInfo()

PDO::errorCode — **Restituisce SQLSTATE associato all'ultima operazione sul database**
Descrizione
PDO pubblico ::codice errore (): ? corda
Elenco parametri:
Questa funzione non contiene alcun parametro.

PDO::errorInfo — Recupera le informazioni sull'errore esteso associate all'ultima operazione sull'handle del database

descrizione
```
public PDO::errorInfo (): array
```

parametri: questa funzione non ha parametri.

```php
<?php
try {
        $pdo->beginTransaction();
        $prepared = $pdo->prepare("INSERT INTO servizi
        (nome,icona,descrizione)
        VALUES
        (:nome, :icona, :descrizione)");
        $prepared->bindParam(':nome', $_REQUEST["nome"]);
        $prepared->bindParam(':icona', $_REQUEST["icona"]);
        $prepared->bindParam(':descrizione',
$_REQUEST["descrizione"]);
        $prepared->execute();
        $pdo->commit();
      }catch (PDOException $e){
        $pdo->rollBack();
        die (message('danger', $e->getMessage()));
      }

echo "\nPDO::errorCode(): ", $pdo->errorCode();
print_r($pdo->errorInfo());

?>
```

L'esempio sopra produrrà:
PDO::codice errore(): 00000

PDO::errorInfo()

Valori di ritorno

PDO::errorInfo() restituisce una matrice di informazioni sull'errore sull'ultima operazione eseguita da questo handle di database. L'array è composto almeno dai seguenti campi:

Elemento	Informazione
0	**Codice di errore SQLSTATE** (un identificatore alfanumerico di cinque caratteri definito nello standard ANSI SQL).
1	**Codice di errore specifico del driver**.
2	**Messaggio di errore specifico del driver.**

```php
<?php
/* Provoke an error --
 bogus SQL syntax */
$stmt = $dbh->prepare(errore sql');
if (!$stmt) {
    echo "\nPDO::errorInfo():\n";
    print_r($dbh->errorInfo());
}
?>
```

L'esempio sopra produrrà:

```
PDO::errorInfo():
Vettore
(
   [0] => HY000
   [1] => 1
   [2] => vicino a "fasullo": errore di sintassi
)
```

PDOStatement::fetch

PDOStatement::fetch — Recupera la riga successiva da un set di risultati

Recupera una riga da un set di risultati associato a un oggetto PDOStatement.

Il parametro determina come PDO restituisce la riga.

Descrizione

```
public PDOStatement::fetch ( int $mode= PDO::FETCH_DEFAULT , int $cursorOrientation= PDO::FETCH_ORI_NEXT , int $cursorOffset= 0 ): misto
```

PDO::FETCH_ASSOC: Return next row as an array indexed by column name Array ([name] => apple [colour] => red)

PDO::FETCH_BOTH: Return next row as an array indexed by both column name and number Array ([name] => banana [0] => banana [colour] => yellow [1] => yellow)

PDOStatement::fetchAll

PDOStatement::fetchAll — Recupera le righe rimanenti da un set di risultati

Descrizione

```
public PDOStatement::fetchAll ( int $mode=
PDO::FETCH_DEFAULT ): array
public PDOStatement::fetchAll ( int $mode=
PDO::FETCH_COLUMN , int $column ): array
public PDOStatement::fetchAll ( int $mode=
PDO::FETCH_CLASS , string $class , ? array $constructorArgs)
: matrice
public PDOStatement::fetchAll ( int $mode=
PDO::FETCH_FUNC , callable $callback ): array
```

```php
<?php
$sth = $dbh-
>prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* Fetch all of the remaining rows in the r
esult set */
print("Fetch all of the remaining rows in t
he result set:\n");
$result = $sth->fetchAll();
print_r($result);
?>
```

Array ([0] => Array ([name] => apple [0] => apple [colour] => red [1] => red) [1] => Array ([name] => pear [0] => pear [colour] => green [1] => green) [2] => Array ([name] => watermelon [0] => watermelon [colour] => pink [1] => pink))

PDO::prepare - update

PDO::prepare — Prepara un'istruzione per l'esecuzione e restituisce un oggetto istruzione

Descrizione

```
public PDO::prepare ( string $query , array $options=[] ): PDOSstatement | falso
```

```
try {
        $pdo->beginTransaction();
          $prepared = $pdo->prepare("UPDATE servizi SET
        `nome` = :nome,
        `icona` = :icona,
        `descrizione` = :descrizione
        WHERE id_servizio = :id_servizio limit 1");

        $prepared->bindParam(':nome', $_REQUEST["nome"]);
        $prepared->bindParam(':icona',
$_REQUEST["icona"]);
        $prepared->bindParam(':descrizione',
$_REQUEST["descrizione"]);
        $prepared->bindParam(':id_servizio',
$_REQUEST["id_servizio"]);

      }catch (PDOException $e){
        $pdo->rollBack();
        die (message('danger', $e->getMessage()));
      }
      $pdo->commit();
```

PDO::prepare select

PDO::prepare — Prepara un'istruzione per l'esecuzione e restituisce un oggetto istruzione

Descrizione

```
public PDO::prepare ( string $query , array $options= [] ): PDOSstatement | falso
```

```
try {
        $prepared = $pdo->prepare("SELECT * FROM tabella WHERE id= :id");
        $prepared->bindParam(':id', $_REQUEST["id"]);
        $prepared->execute();
        $result = $prepared->fetch();

        if($result) {
          //// è un array associativo
        }
      }catch (PDOException $e){
        die (message('danger', $e->getMessage()));
      }
```

PDO::prepare

Prepara un'istruzione SQL da eseguire con il metodo PDOStatement::execute() .
Il modello di istruzione può contenere zero o più indicatori di parametro denominati (:nome) o punto interrogativo (?) per i quali i valori reali verranno sostituiti quando l'istruzione viene eseguita.
È necessario includere un indicatore di parametro univoco per ogni valore che si desidera passare all'istruzione quando si chiama PDOStatement::execute() .
Non è possibile utilizzare un indicatore di parametro denominato con lo stesso nome più di una volta in un'istruzione preparata, a meno che la modalità di emulazione non sia attiva.
La chiamata di PDO::prepare() e PDOStatement::execute() per istruzioni che verranno emesse più volte con valori di parametro diversi ottimizza le prestazioni dell'applicazione consentendo al driver di negoziare la memorizzazione nella cache lato client e/o server del piano di query e metainformazioni.

Inoltre, la chiamata di PDO::prepare() e PDOStatement::execute() aiuta a prevenire gli attacchi di SQL injection eliminando la necessità di citare manualmente ed eseguire l'escape dei parametri.

PDO::preparare //parametri ; valori di ritorno

PARAMETRI

query

Deve essere un modello di istruzione SQL valido per il server di database di destinazione.

options

Questa matrice contiene una o più coppie key=>value per impostare i valori degli attributi per l'oggetto PDOStatement restituito da questo metodo.

È comunemente più utilizzato per impostare il PDO::ATTR_CURSORvalore su PDO::CURSOR_SCROLLper richiedere un cursore scorrevole.

VALORI DI RITORNO

Se il server di database prepara correttamente l'istruzione, PDO::prepare() restituisce un oggetto PDOStatement .

Se il server del database non riesce a preparare correttamente l'istruzione, PDO::prepare() restituisce false o emette PDOException (a seconda della gestione degli errori).

PDO::exec

PDO::exec — Esegue un'istruzione SQL e restituisce il numero di righe interessate

Descrizione
PDO pubblico ::exec (stringa $statement): int | falso

```php
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');

/* Delete all rows from the FRUIT table */
$count = $dbh->exec("DELETE FROM fruit");

/* Return number of rows that were deleted */
print("Deleted $count rows.\n");
?>
```

L'esempio sopra produrrà:

```
Eliminato 1 righe.
```

PDO::exec

DESCRIZIONE

PDO::exec() **esegue un'istruzione SQL in una singola chiamata di funzione,** restituendo il numero di righe interessate dall'istruzione.

PDO::**exec() non restituisce risultati da un'istruzione SELECT**.

PARAMETRI

statement

L'istruzione SQL da preparare ed eseguire.

I dati all'interno della query devono essere correttamente sottoposti a escape .

VALORI DI RITORNO

PDO::exec() restituisce il numero di righe che sono state modificate o eliminate dall'istruzione SQL emessa. Se nessuna riga è stata interessata, PDO::exec() restituisce 0.

PDOSstatement::execute

PDOStatement::execute — Esegue un'istruzione preparata

Descrizione

```
public PDOStatement::execute ( ? array
$params=null ): bollo
```

```php
<?php
/* Execute a prepared statement by binding a variable and value */
$calories = 150;
$colour = 'gre';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour LIKE :colour');
$sth->bindParam('calories', $calories, PDO::PARAM_INT);
/* Names can be prefixed with colons ":" too (optional) */
$sth->bindValue(':colour', "%$colour%");
$sth->execute();
?>
```

PDOSstatement::execute

PARAMETRI

Una matrice di valori con tanti elementi quanti sono i parametri associati nell'istruzione SQL in esecuzione.
Tutti i valori sono trattati come PDO::PARAM_STR.
Non è possibile associare più valori a un singolo parametro; ad esempio, non è consentito associare due valori a un singolo parametro denominato in una clausola IN().
Non è possibile associare più valori di quelli specificati.

VALORI DI RITORNO

Restituisce true in caso di successo o false in caso di fallimento.

PDO::lastInsertId – prepare con array

PDO::lastInsertId — Restituisce l'ID dell'ultima riga inserita o valore della sequenza

Descrizione

```
public PDO::lastInsertId ( ? stringa $name=
null ): stringa | falso
```

```php
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', 'username', 'password');

    $stmt = $dbh->prepare("INSERT INTO test (name, email) VALUES(?,?)");

    try {
        $dbh->beginTransaction();
        $tmt->execute( array('user', 'user@example.com'));
        $dbh->commit();
        print $dbh->lastInsertId();
    } catch(PDOExecption $e) {
        $dbh->rollback();
        print "Error!: " . $e->getMessage() . "</br>";
    }
} catch( PDOExecption $e ) {
    print "Error!: " . $e->getMessage() . "</br>";
}
?>
```

PDO::lastInsertId

PARAMETRI

name

Nome dell'oggetto sequenza da cui deve essere restituito l’ID.

VALORI DI RITORNO

Se non è stato specificato un nome di sequenza per il name parametro, PDO::lastInsertId() restituisce una stringa che rappresenta l'ID riga dell'ultima riga inserita nel database.

Se è stato specificato un nome sequenza per il name parametro, PDO::lastInsertId() restituisce una stringa che rappresenta l'ultimo valore recuperato dall'oggetto sequenza specificato.

Se il driver PDO non supporta questa funzionalità, PDO::lastInsertId() attiva un IM001SQLSTATE.

Le Funzioni di errore - PHP ERROR

Le funzioni di errore vengono utilizzate **per** gestire **la gestione e la registrazione degli errori**.

Ci **consentono di definire regole di gestione degli errori e modificare il modo in cui gli errori possono essere registrati.**

Le funzioni di registrazione ci consentono **di inviare messaggi direttamente ad altre macchine, e-mail o registri di sistema.**

Le funzioni di segnalazione degli errori ci consentono **di personalizzare il livello e il tipo di feedback degli errori fornito.**

error_reporting()

La funzione **error_reporting() specifica quali errori vengono segnalati.**
PHP ha molti livelli di errori e l'uso di questa funzione imposta quel livello per lo script corrente.

```
error_reporting(E_ERROR);
print crypt('ciao');

//non viene stampato Notice: crypt(): No salt
parameter was specified. You must use a randomly
generated salt and a strong hash function
```

```
<?php
// Turn off error reporting
error_reporting(0);

// Report runtime errors
error_reporting(E_ERROR | E_WARNING |
E_PARSE);

// Report all errors
error_reporting(E_ALL);

// Same as error_reporting(E_ALL);
ini_set("error_reporting", E_ALL);

// Report all errors except E_NOTICE
error_reporting(E_ALL & ~E_NOTICE);
?>
```

Eccezioni in PHP

PHP ha un modello di eccezione simile a quello di altri linguaggi di programmazione.

Un'eccezione può essere generata e catturata ("catturata") all'interno di PHP.

Il codice può essere racchiuso in un blocco try, per facilitare la cattura di potenziali eccezioni.

Ogni tentativo deve avere almeno una cattura corrispondente o un blocco finale.

The thrown object must be an instance of the Exception class or a subclass of Exception. Trying to throw an object that is not will result in a PHP Fatal Error.

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(),
"\n";
}

// Continue execution
echo "Hello World\n";
?>
```

ERROR HANDLER

ESEMPIO DI CREAZIONE DI UN PROPRIO ERROR HANDLER

```
set_error_handler("warning_handler",
E_WARNING);

 try {
     $fName='nonesiste.php';
     fopen($fName,'r');
     print "non verrà eseguita \n";
 } catch (Exception $e) {
     printf ("Errore: %s - %s ",$e-
>getCode(),$e->getMessage());
 }

 print "Dopo try catch\n";

 function warning_handler($errno, $errstr) {
     print "warning_handler
{$errno},  {$errstr}";
     die;
 }
```

ESEMPIO DI CHECK E THROW EXCEPTION

```
function checkNum($number) {
    if($number>1) {
      throw new Exception("Value must be 1 or below");
    }
    return true;
  }

  //trigger exception in a "try" block
  try {
    checkNum(2);
    //If the exception is thrown, this text will not be
shown
    echo 'If you see this, the number is 1 or below';
  }
  //catch exception
  catch(Exception $e) {
    echo 'Message: ' .$e->getMessage();
  }
```

PSR-12

A try-catch-finally block looks like the following. Note the placement of parentheses, spaces, and braces.

```
try {
    // try body
} catch (OverflowException $e) {
    // catch body
} catch (PDOException |
OutOfRangeException $e) {
    // catch body
} finally {
    // finally body
}
```

Lista exception:
https://www.php.net/manual/en/class.exception.php

Eccezioni PHP

Un'eccezione è un oggetto che descrive un errore o un comportamento imprevisto di uno script PHP.
Le **eccezioni** vengono **generate da molte funzioni e classi** PHP.
Anche le funzioni e le classi definite dall'utente possono generare eccezioni.
Le **eccezioni sono un buon modo per interrompere una funzione quando incontra dati che non può utilizzare.**
alcuni dei metodi che possono essere utilizzati per ottenere informazioni sull'eccezione:

getCode() //Restituisce il codice di eccezione

getFile() //Restituisce il percorso completo del file in cui è stata lanciata l'eccezione

getLine() // Restituisce il numero di riga della riga di codice che ha lanciato l'eccezione

Eccezioni PHP // getCode()

Il **getCode() restituisce un numero intero** che può essere utilizzato **per identificare l'eccezione.**

```
<?php
try {
  throw new Exception("An error occurred", 120);
} catch(Exception $e) {
  echo "Error code: " . $e->getCode();
}
?>
```

output:

Error code: 120

Eccezioni PHP // getLine()

Il **getLine()** restituisce il numero di riga della riga di codice che ha generato l'eccezione.

```
<?php
try {
  throw new Exception("An error occurred");
} catch(Exception $e) {
  echo $e->getLine();
}
?>
```

Output:

7

COMPOSER

Composer is a tool for dependency management in PHP. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.

Installation - Window

Using the Installer#

This is the easiest way to get Composer set up on your machine.

Download and run Composer-Setup.exe. It will install the latest Composer version and set up your PATH so that you can call composer from any directory in your command line.

composer.json

Questo file descrive le dipendenze del tuo progetto e può contenere anche altri metadati.
In genere dovrebbe andare nella directory più in alto del tuo progetto/repository VCS (root)

require key

La prima cosa che specifichi in composer.json è la chiave require. Stai dicendo a Composer da quali pacchetti dipende il tuo progetto.

require prende un oggetto che associa i nomi dei pacchetti (ad es. monolog/monolog) ai vincoli di versione (ad es. 1.0.*).

Il compositore utilizza queste informazioni per cercare il set di file corretto nei "repository" del pacchetto che l'utente registra utilizzando la chiave dei repository o in Packagist.org,

```
{
    "require": {
        "monolog/monolog": "2.0.*"
    }
}
```

Package names

Il nome del pacchetto è costituito dal nome del fornitore (VENDOR) e dal nome del progetto.

Nel nostro esempio, stiamo richiedendo il pacchetto Monolog con il vincolo di versione 2.0.*. Ciò significa qualsiasi versione nel ramo di sviluppo 2.0 o qualsiasi versione maggiore o uguale a 2.0 e inferiore a 2.1 (>=2.0 <2.1).

```
{
    "require": {
        "monolog/monolog": "2.0.*"
    }
}
```

composer init

Consente di creare in modalità guidata un composer.json.
Quando esegui il comando, ti chiederà in modo interattivo di compilare i campi, utilizzando alcune impostazioni predefinite intelligenti.

Minumum stability: Must be empty or one of: stable, RC, beta, alpha, dev

```
PS C:\xampp\htdocs\its\corso-magento-test\composer> composer init

Welcome to the Composer config generator

This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [mauro/composer]: soluzione-software/php-sample
Description []: progetto di esempio
Author [mauro <mauro.casadei@soluzionesoftware.com>, n to skip]:

Minimum Stability []: dev
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []: MIT

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? NO
Would you like to define your dev dependencies (require-dev) interactively [yes]? NO
Add PSR-4 autoload mapping? Maps namespace "SoluzioneSoftware\PhpSample" to the entered relative
path. [src/, n to skip]:

{
    "name": "soluzione-software/php-sample",
    "description": "progetto di esempio",
    "type": "project",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "SoluzioneSoftware\\PhpSample\\": "src/"
        }
    },
    "authors": [
        {
            "name": "mauro",
            "email": "mauro.casadei@soluzionesoftware.com"
        }
    ],
    "minimum-stability": "dev",
    "require": {}
}
```

composer install

Il comando install legge il file composer.json dalla directory corrente, risolve le dipendenze e le installa nel fornitore.

php composer.phar install

Se è presente un file composer.lock nella directory corrente, utilizzerà le versioni esatte da lì invece di risolverle. Ciò **garantisce che tutti coloro che utilizzano la libreria ottengano le stesse versioni delle dipendenze**.

Se non è presente alcun file composer.lock, Composer ne creerà uno dopo la risoluzione delle dipendenze.

composer update

Per ottenere le ultime versioni delle dipendenze e aggiornare il file composer.lock, dovresti usare il comando update.
Questo comando è anche chiamato upgrade poiché fa lo stesso di upgrade se stai pensando a apt-get o gestori di pacchetti simili.

```
> composer update
```

Se vuoi aggiornare solo alcuni pacchetti e non tutti, puoi elencarli come tali:

```
> composer update vendor/package
```

Questo risolverà tutte le dipendenze del progetto e scriverà le versioni esatte in composer.lock.

ATTENZIONE: NON è OPPORTUNO AGGIORNARE TUTTE LE DIPENDENZE SENZA PRIMA TESTARLE ALTRIMENTI SI RISCHI DI INCORRERE IN QUALCHE INCOMPATIBILITÀ

QUINDI MEGLIO AGGIORNARE SOLO I PACCHETTI DI CUI SI È CERTI

autoload.php

L'autoload.php è un file che consente il caricamento delle librerie semplicemente venendo incluso nel nostro codice. Basta un singolo *require*:

```
require __DIR__ . '/vendor/autoload.php';
```

composer remove

Utilizzato per rimuovere un pacchetto sia dal composer.json che dalla vendor e aggiornare composer.lock

```
> composer remove symfony/google-mailer
```

PSR-4 Standard autoloading

PSR-4 costituisce un nuovo standard per gestire l'autoloading e fa uso dei namespace, tecnica che evita i conflitti tra classi con lo stesso nome, ma appartenenti a librerie diverse.
Per poter gestire le nostre classi mediante PSR-4, immaginiamo lo scenario seguente:

```
cartella-sito /
        sotto-cartella-mie-classi/
           classe1.php
           classe2.php
           classe3.php
```

Assegniamo alle classi un namespace, uguale per tutte:

```
<?php

namespace MiaAzienda;

class Classe1 {
        public function __construct()
        {
                echo "Ciao, sono classe1.";
        }
}

In composer.json
{
        "autoload": {
                "psr-4": {
                        "MiaAzienda\\":"sotto-cartella-mie-
classi/"
                }
        }
}
```

Poi
> composer dump-autoload –o

Nel file (es index.php):

```
<?php
        require "vendor/autoload.php";

           use MiaAzienda\Classe1;
           use MiaAzienda\Classe2;
           use MiaAzienda\Classe1;

           $obj1 = new Classe1();
           $obj2 = new Classe2();
           $obj3 = new Classe3();
```

Esempio con symfony/mailer e composer

composer require symfony/mailer

composer require symfony/mime

Nell'esempio utilizzo mailtrap
smtp://*******user****:***pwd****@smtp.mailtrap.io:2525/?encryption=ssl&auth_mode=login

elenco smtp previsti
https://symfony.com/doc/current/mailer.html

File: index.php

```php
<?php
    require __DIR__ . '/vendor/autoload.php';

    use Symfony\Component\Mailer\Transport;
    use Symfony\Component\Mailer\Mailer;
    use Symfony\Component\Mime\EMail;

    $transport = Transport::fromDsn('smtp://*******:*******@smtp.mailtrap.io:2525/?encryption=ssl&auth_mode=login');
    $mailer = new Mailer($transport);

    $email = (new Email())
        ->from('docentemaurocasadei@gmail.com')
        ->to('docentemaurocasadei@gmail.com')
        //->cc('*********')
        //->bcc('**********')
        //->replyTo('************')
        //->priority(Email::PRIORITY_HIGH)
        ->subject('Time for Symfony Mailer!')
        ->text('Sending emails is fun again!')
        ->html('<p>See Twig integration for better HTML integration!</p>');

    $mailer->send($email);
    echo "Mail Inviata";
```

Classe Carbon – gestire le Date

The Carbon class is inherited from the PHP DateTime class.

https://carbon.nesbot.com/docs/

consente di gestire in modo più semplice le date, per utilizzarlo da composer:

```
composer require nesbot/carbon
```

nell'index.php

```
<?php require 'vendor/autoload.php'; use Carbon\Carbon; printf("Now: %s", Carbon::now());
```

PSR-1

Imposta gli standard per la programmazione

1. Files MUST use only <?php and <?= tags.
2. <?= equivale a <?php echo
3. Files MUST use only UTF-8 without BOM for PHP code.
   (con BOM ha 3 caratteri **EF BB BF all'inizio**)
4. Nel file non dovrebbero essere presenti sia (classes, functions, constants, etc.) che side-effects (e.g. generate output, change .ini settings, etc.)
5. **each class is in a file by itself**, and is in a namespace of at least one level: a top-level vendor name
6. Class names = **PascalCase**
7. **Class constants** UPPER_CASE
   [upper case with underscore separators.]
8. **Method names** = **camelCase**

https://www.php-fig.org/psr/psr-12/

Ex 3. cosa sarebbe opportuno **NON** fare

```
// side effect: change ini settings ini_set('error_reporting', E_ALL);
// side effect: loads a file
include "file.php";
// side effect: generates output
echo "<html>\n"; // declaration
function foo() { // function body }
```

Ex.3 Cosa Fare:

```
<?php
// declaration
function foo()
{
    // function body
}
// conditional declaration is *not* a side effect
if (! function_exists('bar')) {
    function bar()
    {
        // side effect: loads a file
                include "file.php";
    }
}
```

In questo modo non viene "sempre" eseguita ma solo quando viene chiamata la funzione

PSR-12

4.3 Properties and Constants

Visibility MUST be declared on all properties.

Visibility MUST be declared on all constants if your project PHP minimum version supports constant visibilities (PHP 7.1 or later).

```php
<?php namespace

Vendor\Package;

class ClassName {
    public const NOME_COSTANTE = 'valore';
    public $foo = null;
    public static int $bar = 0;
}
```

PSR-12

Methods and Functions

Visibility MUST be declared on all methods.

Method names MUST NOT be prefixed with a single underscore to indicate protected or private visibility.
That is, an underscore prefix explicitly has no meaning.

In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

```
namespace Vendor\Package;

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

Funzione PHP mail()

La funzione **mail()**
ti **per**mette di **inviare email** direttamente **da** uno **script**.
Sintassi:
mail(to,subject,message,headers,parameters);
Valore di ritorno: Restituisce il valore hash del parametro address o FALSE in caso di errore.

```
<?php
// the message
$msg = "First line of text\nSecond line of text";

// use wordwrap() if lines are longer than 70 characters
$msg = wordwrap($msg,70);

// send email
mail("someone@example.com","My subject",$msg);
?>
```

da preparare come esempio

creazione esempio con logica MVC
1 File che riceve la chiamata e i parametri
1 File che gestisce il database
1 File che gestisce la vista

MVC è un design pattern utilizzato per separare i dati (Model), le interfacce utente (View) e la logica dell'applicazione (Controller)

Creiamo le cartelle di base per il vostro MVC:

1.app
2.config
3.public
4.views
5.routes