IL PHP E MYSQL

Aggiornato al 3 aprile 2023

Per gli studenti:

- dei corsi SIA
- dell'Itis Informatica e Telecomunicazioni
- dell'Università Informatica, Ingegneria gestionale, Ingegneria biomedica, Politecnico.

Fonte orginale: https://guidaphp.it/base

Esercizi fonte originale: https://www.allphptricks.com/

INDICE DELLA GUIDA

Introduzione a PHP pag. 4
- **Che cos'è il PHP**

Installazione pag. 7
- **Installare PHP su Windows con XAMPP**
- **Configurare PHP**

Sintassi PHP pag. 64

Variabili e tipi di dati pag. 68
- **Tipi di dati**
- **Boolean**
- **Integer**
- **Float**
- **String**
- **Costanti**

Array pag. 82
- **Array associativi**
- **Array multidimensionali**
- **Ordinamento array**

Operatori pag. 103
- **Di incremento/decremento**
- **Di stringa**
- **Aritmetici**
- **Di assegnazione**
- **Di confronto**
- **Logici**
- **Bitwise**

Strutture di controllo pag. 113
- **Istruzioni condizionali**
- **Cicli iterativi**
- **Costrutti include e require**
- **Costrutto return**

Funzioni pag. 133
- **Funzioni anonime**
- **Funzioni e tipi di dati in PHP 7**
- **Funzioni per gestire array**
- **Funzioni per gestire stringhe**

Variabili superglobali pag. 185
- **$_COOKIE**
- **$_SESSION**
- **$_FILES**

File e directory pag. 210
- Funzioni di gestione file
- File CSV
- File JSON
- File XML

PHP e MySQL pag. 252

Cos'è un database
DBMS e MySQL
Come funziona MySQL
Il linguaggio SQL
Creare un database
Creare un tabella
Inserire i dati
Modificare i dati
Selezionare i dati
PHP e MySQL

Estensione MySQLi pag. 260

Cos'è MySQLi
Connessione al database
Chiusura della connessione
La funzione mysqli_query
Creazione database - CREATE DATABASE
Creazione tabelle - CREATE TABLE
Inserimento dati - INSERT INTO
Estrazione dati - SELECT
Aggiornamento dati - UPDATE
Cancellazione dati - DELETE

PDO pag. 270

Esercizii completi Php e Mysqli

- Login e registrazione con Php e MySQLi (ultima versione) con file zippato della procedura **pag. 278**

- Inserisci, visualizza, modifica ed elimina record dal database utilizzando PHP e MySQLi e file zippato **pag. 292**

Cos'è PHP: introduzione al linguaggio

PHP è un **linguaggio di programmazione** creato da Rasmus Lerdorf nel lontano 1995 per la creazione di pagine web dinamiche. Da allora si è evoluto in un linguaggio completo e professionale, grazie anche al supporto della community che ruota intorno ad esso.

L'acronimo PHP sta per *Hypertext Preprocessor*, mentre nella prima implementazione del linguaggio stava per *Personal Home Pages*.

Con la sua **semplicità e flessibilità**, PHP è adatto sia per lo sviluppo di un semplice blog che di applicazioni web più complesse, come ad esempio un social network o un forum.

Nel corso degli anni sono nati numerosi software open source scritti in PHP, tra i quali sono da menzionare WordPress, Joomla!, Magento, Prestashop tra i **CMS** e Symfony, Laravel e **CodeIgniter** tra i **framework**.

Tra le funzionalità o applicazioni che è possibile implementare con PHP troviamo:

- invio modulo di contatto
- upload di file
- registrazione e login area riservata
- mailing list
- gestione file
- blog
- Content Management Systems (CMS)
- Customer Relationship Management (CRM)
- piattaforme di e-learning
- piattaforme di e-commerce
- e molto altro

Dopo questa breve introduzione analizziamo le principali caratteristiche e funzionalità del linguaggio, vediamo un esempio di codice PHP e perché oggi vale la pena imparare questo linguaggio.

Indice dei contenuti

1. **Caratteristiche**
2. **Esempio di codice**
3. **Funzionalità**
4. **Licenza PHP**
5. **Perché imparare PHP?**

Caratteristiche

Elenchiamo alcune delle principali **caratteristiche** di PHP:

- È un **linguaggio di scripting interpretato**.
- È un **linguaggio server-side**: il codice PHP viene eseguito sul server web dall'interprete del linguaggio il quale converte il codice PHP in HTML e lo invia al client, ossia il browser dell'utente.
- La sua **sintassi** è derivata dal linguaggio C e dal Perl.
- È un linguaggio **HTML-embedded**, in quanto è possibile inserire codice PHP all'interno di una pagina contenente codice HTML.

- Nasce come **linguaggio a tipizzazione debole**, infatti è possibile assegnare alla stessa variabile differenti **tipi di dati**.
- Si integra perfettamente con i più diffusi DBMS tra cui **MySQL**, PostgreSQL, MariaDB, MSSQL.
- Consente l'interfacciamento a numerose librerie: cURL, GD, OpenSSL, etc...
- Ha un semplice meccanismo di garbage collection, che si occupa di liberare la memoria utilizzata al termine dell'esecuzione di uno script.
- È supportato dai **più diffusi web server**, primi su tutti Apache e nginx, ma anche Microsoft-IIS e Litespeed.

Esempio di codice

Ma come è strutturato uno programma (o script) scritto in linguaggio PHP?

Ecco il più classico degli esempi che troviamo in tutti i libri di programmazione

```
<?php

echo "Hello World!";

?>
```

L'esempio appena visto è un semplice programma PHP che stampa a video la scritta Hello World!. Il codice racchiuso tra i tag <?php e ?> viene eseguito dall'interprete PHP e il codice HTML risultante viene inviato al browser dell'utente.

In realtà PHP è in grado di creare applicazioni ben più complesse e potenti di una semplice stampa a video, ma questo lo vedremo nelle successive guide quando andremo ad analizzare le caratteristiche del linguaggio.

Funzionalità

Le novità più importanti del linguaggio sono state introdotte a partire da PHP 5, con il miglioramento del supporto alla programmazione orientata agli oggetti (OOP) e l'introduzione di PHP Data Objects (PDO), una libreria per l'accesso ai **database** tramite un'unica interfaccia.

Tuttavia è con l'uscita della versione 7, avvenuta alla fine del 2015, che il linguaggio ha fatto un notevole passo avanti rispetto al passato. Oltre a riscrivere il **motore del linguaggio**, rendendo **PHP il più veloce tra i linguaggi di scripting**, gli sviluppatori hanno introdotto numerose nuove funzionalità che consentono la scrittura di codice più robusto e sicuro.

Alcune delle **funzionalità introdotte con PHP 7** sono elencate di seguito:

1. Dichiarazione dei **tipi di dati in ingresso e in uscita per le funzioni**.
2. Operatori *null coalesce* e *spaceship*.
3. Definizione di costanti di tipo array tramite la funzione define().
4. Possibilità di creare classi anonime.

Con l'uscita di **PHP 7.4** a fine novembre 2019, sono previste nuove funzionalità e un ulteriore miglioramento delle performance di esecuzione.

Da segnalare la possibilità di assegnare il tipo di dato per le proprietà di classe.

Licenza PHP

Quando un nuovo software viene rilasciato, questo di solito comprende anche una licenza d'uso, che ne determina le modalità specifiche per l'utilizzo e la ridistribuzione.

Nel caso di PHP si è scelto di rilasciarlo sotto una specifica licenza open source, denominata ***PHP license***, che comprende alcune restrizioni sul nome "PHP".

Con questa licenza è possibile ridistribuire il codice sorgente e i file binari del linguaggio alle seguenti condizioni:

- La PHP License deve essere sempre inclusa.
- Il termine "PHP" non deve essere usato per promuovere software derivati da PHP.
- Il seguente riconoscimento deve essere sempre presente
  "This product includes PHP software, freely available from <http://www.php.net/software/>".

Perché imparare PHP?

La semplicità d'utilizzo data da una curva di apprendimento bassa, ha reso PHP uno dei linguaggi di programmazione più conosciuti e utilizzati. Basti pensare che è **utilizzato da quasi l'80%** di tutti i siti internet del mondo.

Alcuni motivi per cui vale la pena imparare PHP:

- È un linguaggio **open source**.
- Ha un ampio supporto della community.
- Esistono migliaia di **packages** ben testati e supportati, che svolgono svariate funzionalità.
- Bassa curva di apprendimento rispetto ad altre linguaggi come ASP, JSP, Perl.
- Praticamente tutti i provider di hosting forniscono pieno supporto alle ultime **versioni di PHP**.
- Le novità introdotte con la versione 7 lo hanno reso un linguaggio di programmazione completo.
- Essendo un linguaggio lato server, è necessario solamente installarlo su un server ed è sufficiente un browser web per visualizzare pagine scritte in PHP.
- È facilmente configurabile tramite un semplice file di testo, chiamato ***php.ini***.
- Ha il supporto integrato con i più diffusi database: MySQL, MSSQL, PostgreSQL.
- È un linguaggio multipiattaforma: questo significa che è possibile sviluppare la propria applicazione web su diversi sistemi operativi (Windows, Linux o Mac).

Come già detto, PHP è un linguaggio interpretato, ossia esiste un programma detto interprete che si occupa di tradurre il codice PHP in linguaggio comprensibile ai computer. Nella guida successiva parleremo proprio di questo interprete e del suo funzionamento.

Installare un ambiente PHP

Nelle prossime guide vedremo come **installare e configurare un ambiente di lavoro** sul proprio computer per poter eseguire programmi PHP.

Ciò di cui abbiamo bisogno è un web server compatibile con PHP che gestisca le richieste del client, dell'ultima **versione di PHP** e di un **database** per poter memorizzare i dati in maniera permanente su disco.

Quindi, prima di iniziare a scrivere un **programma PHP**, dobbiamo installare sul computer:

- Il web server Apache
- PHP ultima versione
- Il database MySQL

Anche se è possibile installarli singolarmente, per comodità è preferibile scegliere un pacchetto preconfigurato, che comprende tutti i software che ci servono. Il più diffuso è XAMPP, facile da installare e configurare.

XAMPP è un ambiente di sviluppo web disponibile per Windows, Linux e Mac. Consente di creare applicazioni web con Apache, PHP e MySQL. Fornisce inoltre PhpMyAdmin, lo strumento di amministrazione per gestire facilmente i database MySQL utilizzando un browser web.

La guida successiva descrive passo passo **come installare XAMPP** su un computer con sistema operativo Windows 7. Data la sua semplicità, la procedura d'installazione dovrebbe essere pressoché identica anche per le versioni di Windows 8.1 e 10.

Installare PHP su Windows con XAMPP

Questa guida descrive la procedura per **installare un ambiente di lavoro PHP su Windows**. Allo scopo verrà utilizzato il software gratuito XAMPP che installeremo su Windows 7.

Diamo una breve descrizione di XAMPP e poi iniziamo la procedura di installazione.

Cos'è XAMPP

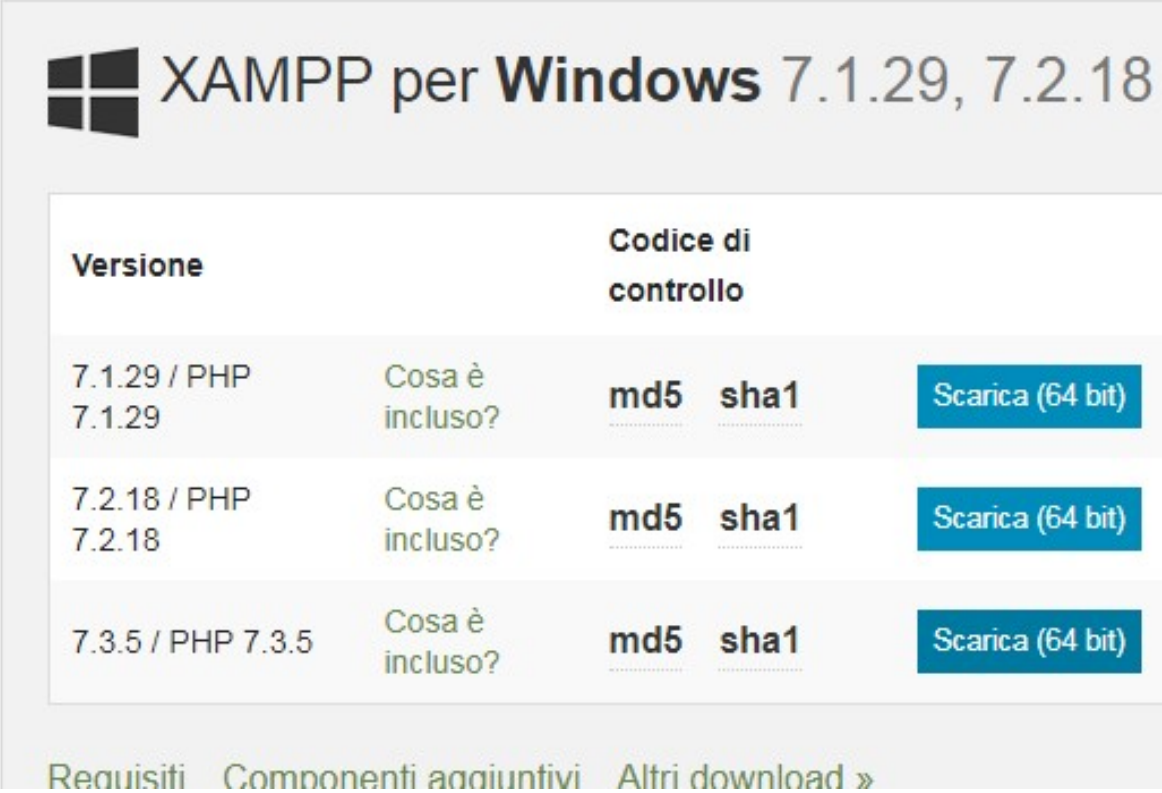
XAMPP è un ambiente di **sviluppo PHP** open source, semplice da installare e utilizzare, disponibile per Windows, Linux e Mac.

Oltre ai software Apache, PHP e MySQL, il pacchetto di installazione comprende phpMyAdmin per la gestione dei **database MySQL** e altre applicazioni per lo sviluppo web.

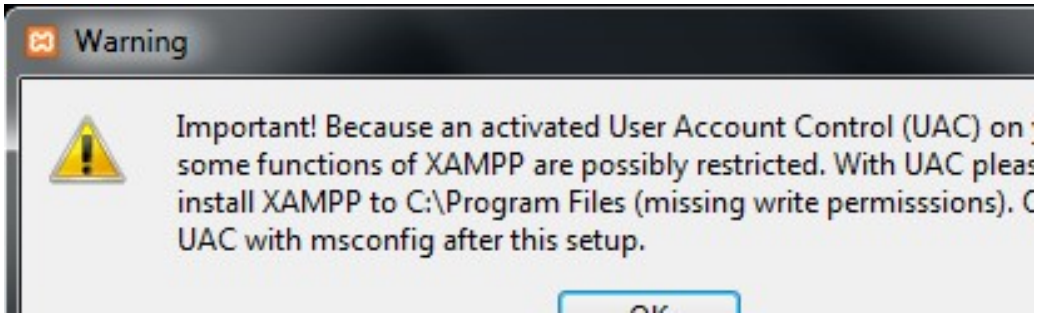
Installare XAMPP su Windows 7, 8, 10, 11

Vediamo i passi per installare XAMPP su Windows 7:

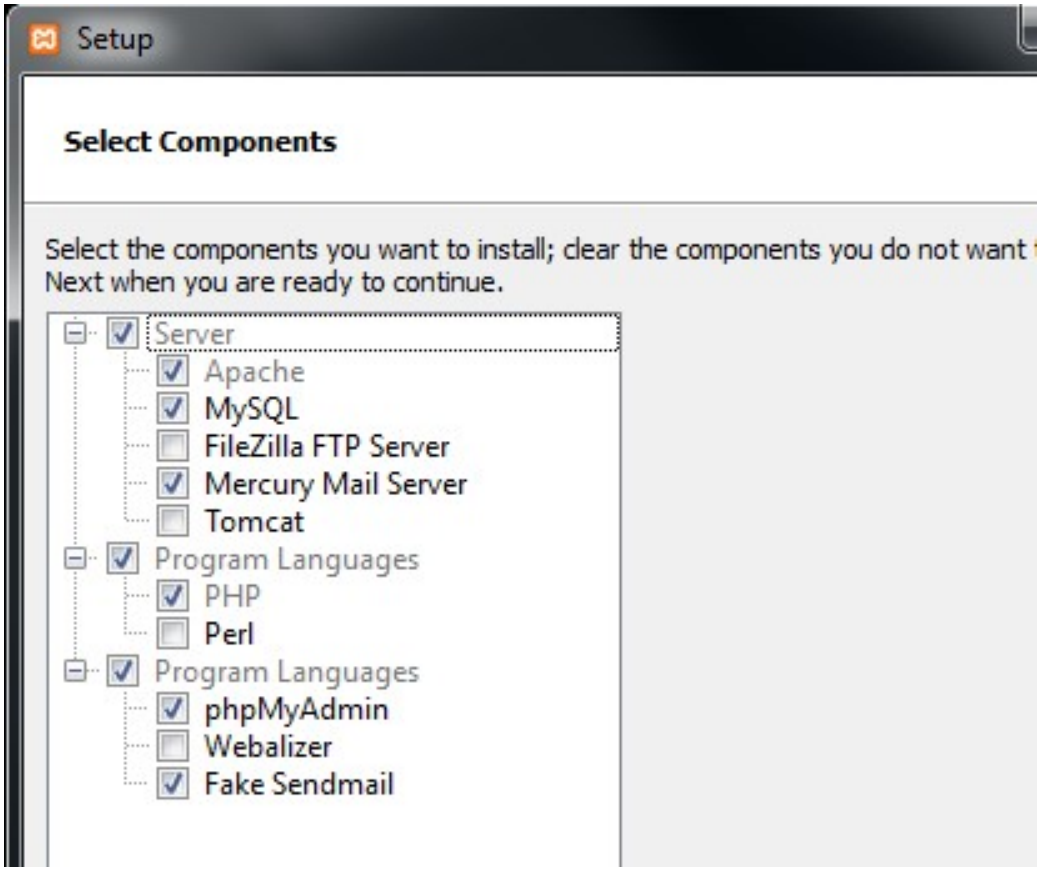
1. Scarica il pacchetto di installazione con l'ultima release di PHP da **qui**. Fai partire l'installazione con un doppio clic sul file scaricato.



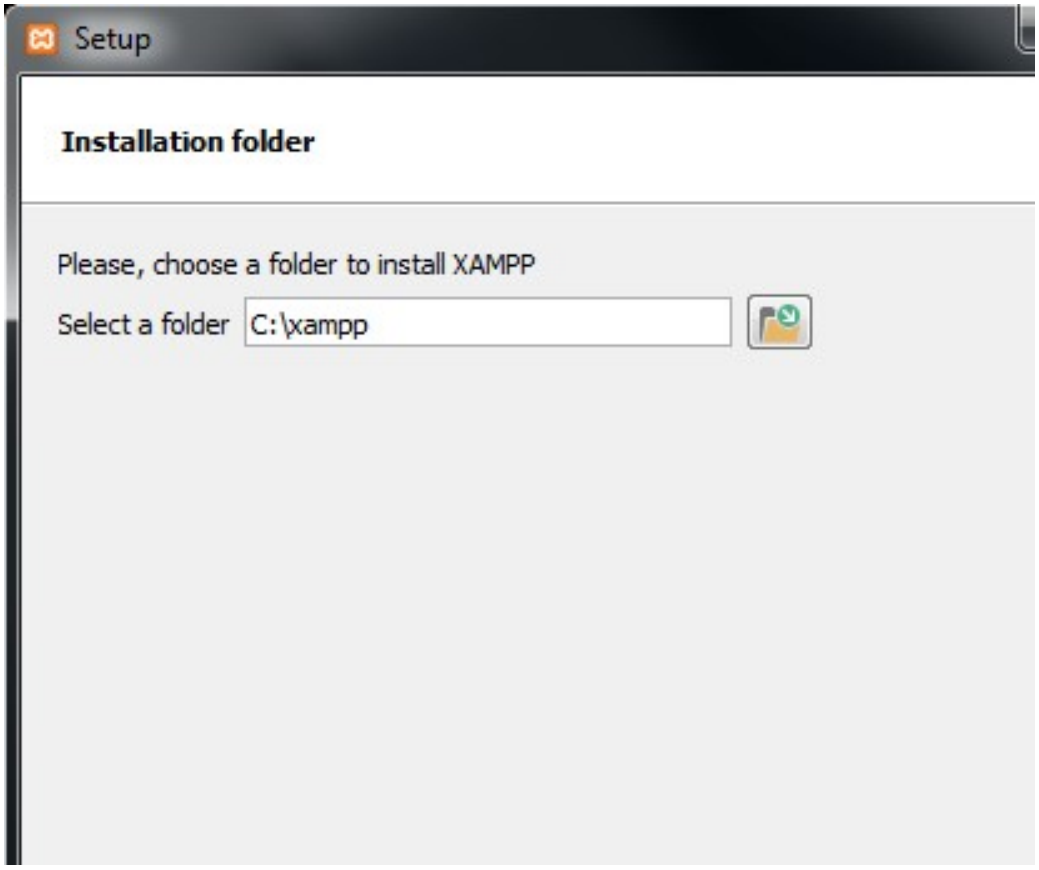
2. Clicca OK quando compare la finestra di avviso mostrata in basso.



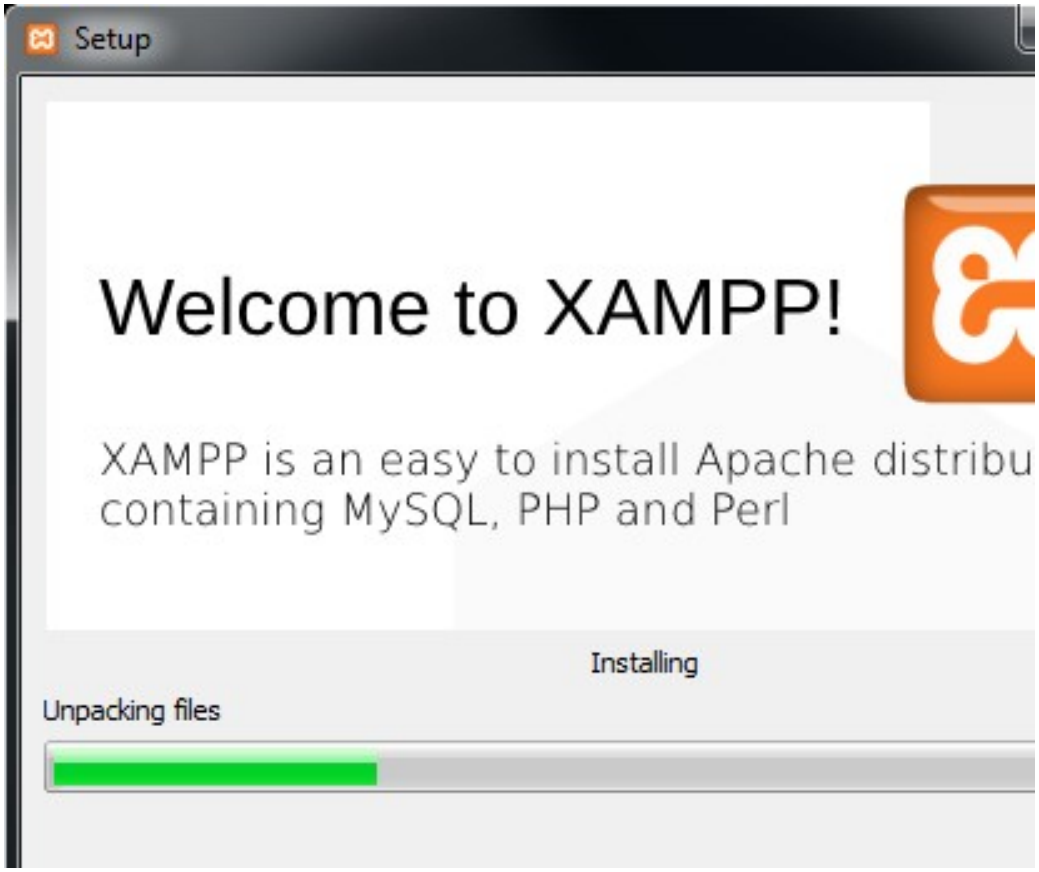
3. Nella schermata seguente seleziona i componenti da installare. Apache, MySQL, PHP e phpMyAdmin. Il resto non serve a parte Mercury Mail Server e Fake Sendmail che potrebbero tornare utili per simulare l'invio e la ricezione di email. Clicca su Next.



4. Nella schermata "Cartella di installazione" modifica il percorso da C:\Program Files (x86) a C:\xampp e clicca su Next.

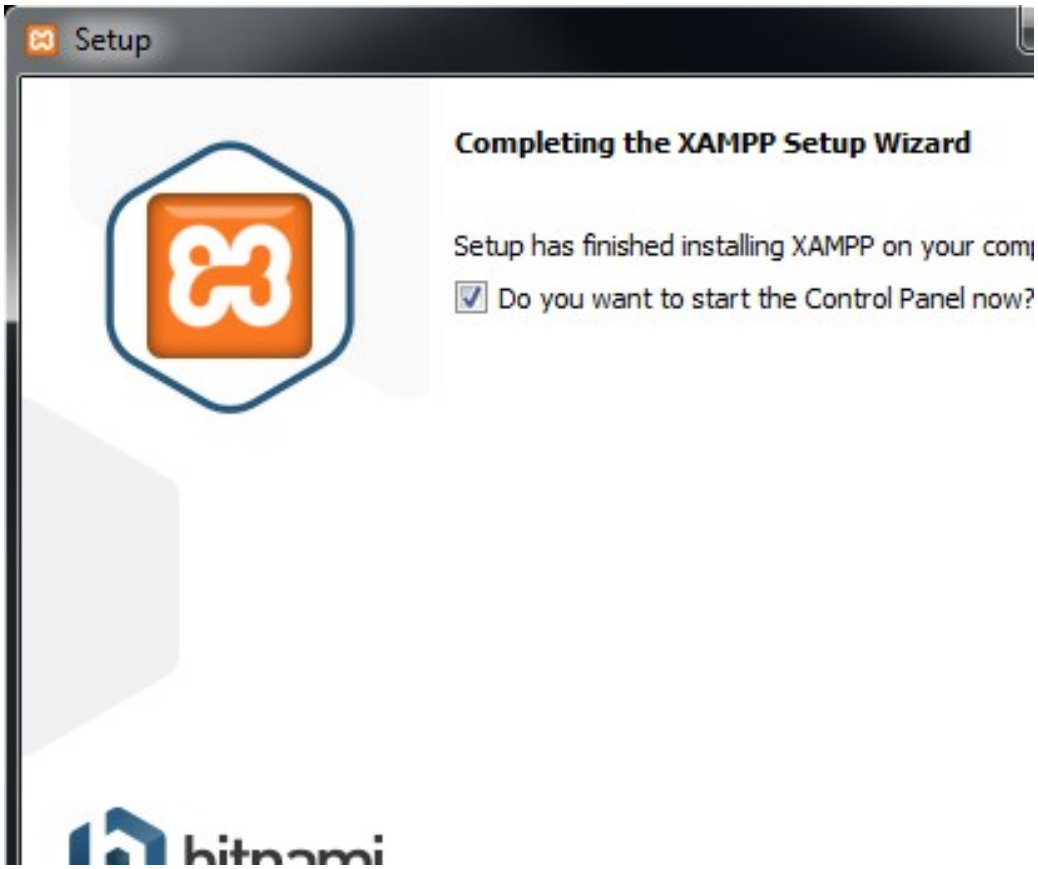


5. Attendi che i pacchetti vengano scaricati e installati sul tuo computer.

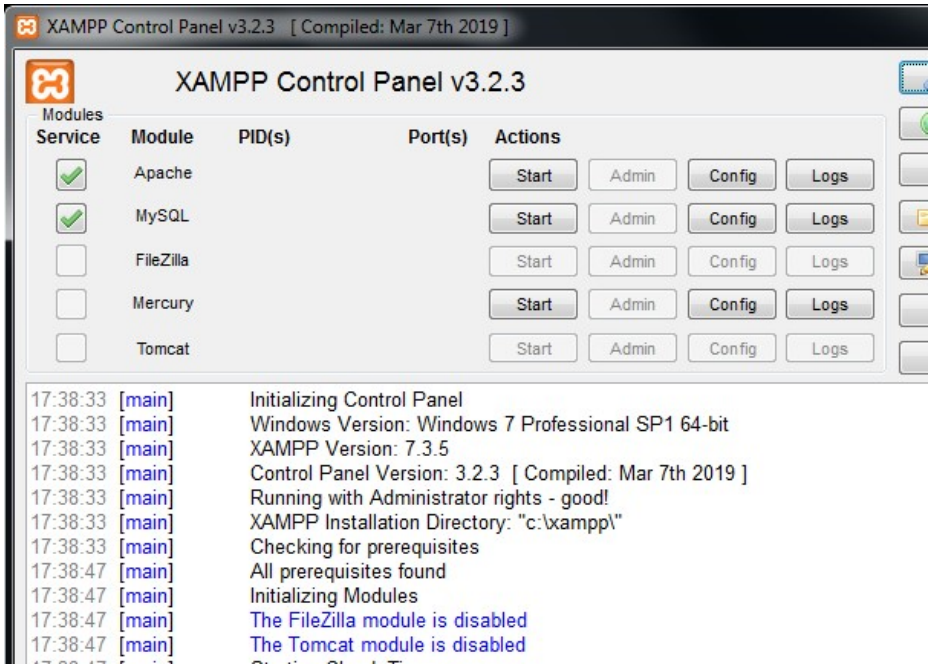


6. Se il firewall del computer dovesse bloccare l'esecuzione di Apache, clicca su "Consenti accesso".

7. Nella schermata di fine installazione tieni spuntata l'opzione "Vuoi avviare il pannello di controllo adesso?".

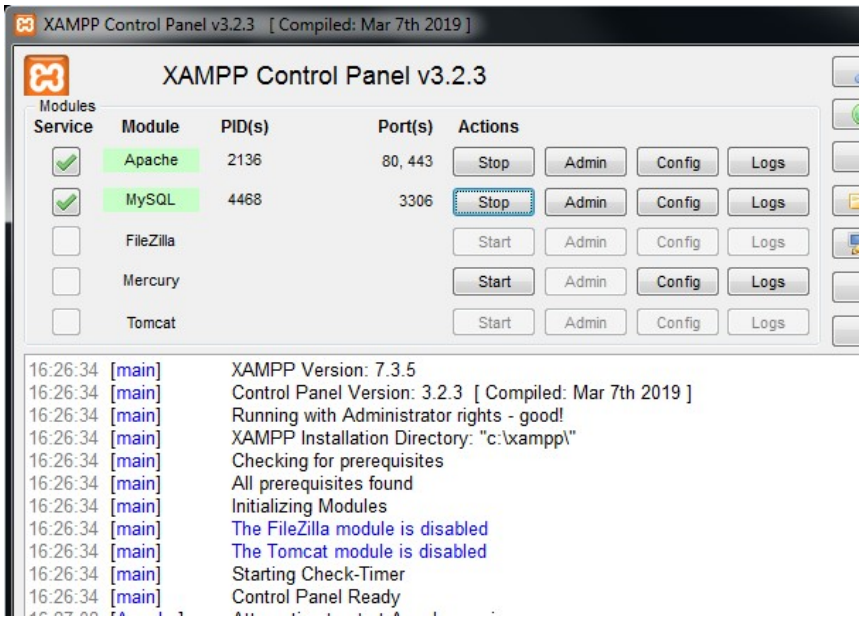


8. Seleziona la lingua di preferenza (inglese o tedesco).
9. Se l'installazione è andata a buon fine, dovresti visualizzare il pannello di controllo con i moduli Apache e MySQL registrati come servizi di Windows (spunta verde sotto "Service").



10. Clicca sul pulsante "Start" per avviare Apache e fai lo stesso per MySQL.

N.B Se compare la finestra di avviso sul "Controllo account utente" clicca OK. Se non vuoi che venga mostrata ogni qualvolta Apache si avvia, devi aprire il pannello di controllo come amministratore (Tasto destro sul file C:\xampp\xampp-control.exe e clic su "Esegui come amministratore").



11. Apri il tuo browser preferito e visita http://localhost.

Dovresti visualizzare la schermata di benvenuto di XAMPP, il che vuol dire che Apache è stato installato ed è funzionante.

Pannello di controllo di XAMPP

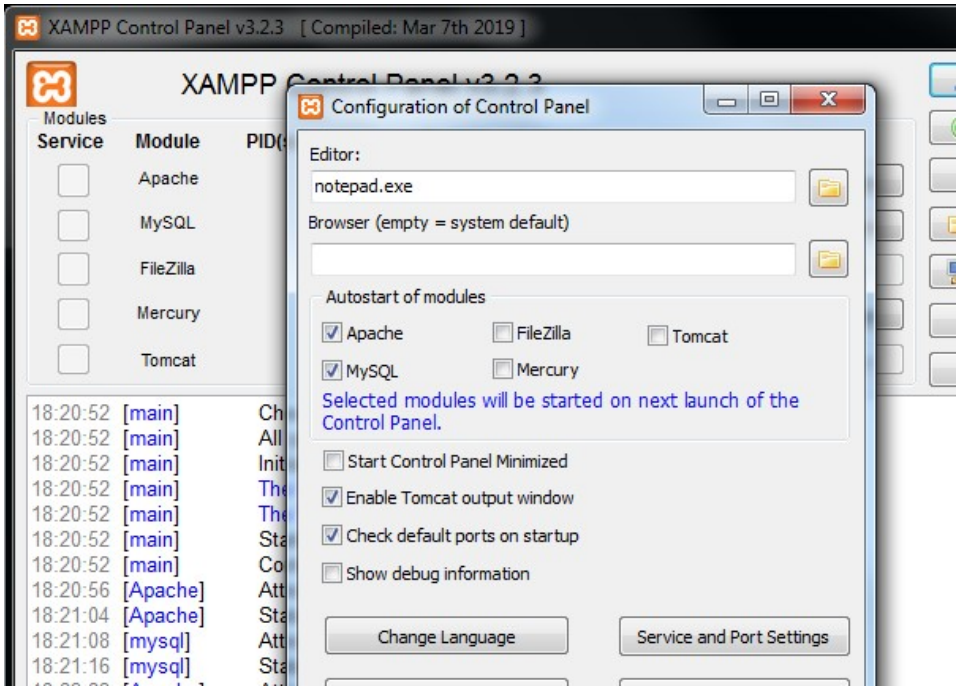
Vediamo velocemente i comandi dell'interfaccia del pannello di controllo.

Per ogni modulo è possibile visualizzare:

- Se il modulo è avviato (spunta verde) oppure no (in caso possiamo avviarlo cliccando sull'icona con la X rossa).
- Il nome del modulo (Apache, MySQL, etc...).
- Il PID che il sistema operativo ha assegnato al processo.
- La porta in ascolto, modificabile da "Config" => "Service and Port Settings".
- Diversi pulsanti per consentire varie azioni sui moduli: "Start/Stop" per avviare/stoppare il modulo, "Admin" per amministrarlo, "Config" per cambiare i settaggi e "Logs" che visualizza i file di log dove vengono salvati eventuali errori e le cause di tali errori.

Sono presenti anche alcune utili funzionalità:

- **Config**: serve a configurare alcune impostazioni. Ad esempio possiamo spuntare le voci "Apache" e "MySQL" per poterli avviare automaticamente quando eseguiamo il pannello di controllo. Oppure modificare la porta in ascolto per i moduli.



- **Netstat**: fornisce una panoramica dei processi (nome, IP, porta e PID) in esecuzione che utilizzano connessioni TCP.
- **Shell**: apre il terminale di sistema di Windows per lanciare comandi.
- **Explorer**: apre la directory dove è installato XAMPP C:\xampp
- **Services**: esegue il comando services.msc che apre la schermata dei servizi attivi sul sistema operativo e che possono essere avviati/arrestati.
- **Help**: apre una schermata che consente di consultare il file "Readme" o chiedere assistenza sul forum.
- **Quit**: chiude il pannello di controllo, stoppando i moduli attivi fino a quel momento.

Verificare l'installazione di PHP

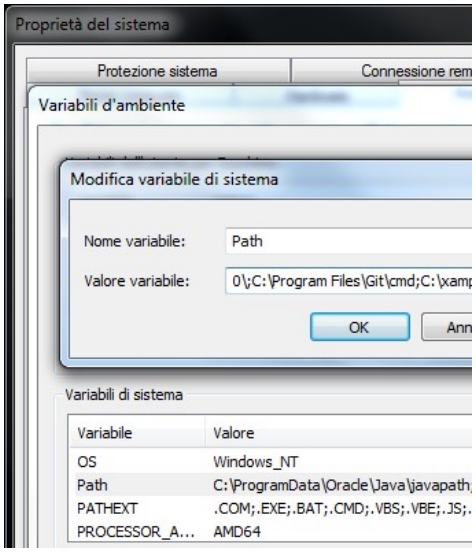
È possibile verificare l'installazione di PHP in due modi:

1. Visita la pagina http://localhost e clicca su PHPinfo in alto a destra. Ti saranno mostrati info e settaggi di PHP. Nella **guida configurazione PHP** vedremo dove vengono prese queste informazioni e come possiamo modificarle.
2. Il secondo metodo richiede un pò di passaggi in più: apri il prompt dei comandi ("Start" => "Esegui", digita il comando cmd e premi invio) e digita il comando php -v.
   Otterrai il seguente errore: "php non è riconosciuto come comando interno o esterno, un programma eseguibile, o un file batch."

   Cosa vuol dire?

   Quando in Windows lanciamo un comando, il sistema operativo cerca il file .exe associato al comando digitato all'interno delle **variabili d'ambiente**. È possibile inserire il percorso dell'eseguibile PHP (php.exe) tra queste variabili, in modo tale Windows sappia sempre dove andare a cercare l'interprete PHP per eseguire comandi direttamente dal prompt (nel nostro caso vogliamo eseguire php -v).

   Per aggiungere l'eseguibile PHP clicca su "Start" => "Esegui", digita il comando sysdm.cpl e premi invio, clicca su "Avanzate" e poi su "Variabili d'ambiente". Scorri tra le *variabili di sistema* fino a trovare path, fai doppio clic e posizionati alla fine della stringa nel campo di testo, aggiungi la seguente stringa ;C:\xampp\php e dai l'OK (non dimenticare il punto e virgola iniziale).



   Adesso possiamo eseguire il nostro comando e la figura seguente mostra appunto l'output di php -v, da cui possiamo vedere la versione di PHP in uso sul sistema, la data di compilazione e altre informazioni come la versione del motore PHP (lo Zend Engine).

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versione 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tutti i diritti riservat

C:\Users\Zucchino>php -v
PHP 7.3.5 (cli) (built: May  1 2019 13:17:17) ( ZTS MSVC15 (Visual
 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.5, Copyright (c) 1998-2018 Zend Technologies

C:\Users\Zucchino>_
```

Ora che hai tutto il necessario per testare il codice degli esempi, passiamo alla successiva guida in cui parleremo di **come configurare PHP**.

Configurare PHP: il file php.ini

Dopo aver **installato PHP tramite XAMPP**, nella directory C:\xampp\php è possibile trovare un file di testo di nome php.ini, che ci consente di **intervenire sulla configurazione PHP** e le funzionalità del linguaggio.

Nel caso PHP sia eseguito come modulo del server, il file viene letto una sola volta (all'avvio del server). Mentre nel caso PHP sia eseguito in modalità CGI o CLI, il file viene letto ad ogni esecuzione.

Indice dei contenuti

1. **Direttive php.ini**
2. **Estensioni del linguaggio**
3. **Verificare la configurazione PHP**

Direttive php.ini

Il settaggio della configurazione avviene tramite la modifica delle cosiddette ***direttive*** presenti all'interno del file, di cui è mostrato un esempio:

```
[PHP]

;;;;;;;;;;;;;;;;;;;

; About php.ini   ;

;;;;;;;;;;;;;;;;;;;

; PHP's initialization file, generally called php.ini, is responsible for

; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a number of locations.

; The following is a summary of its search order:

; 1. SAPI module specific location.

; 2. The PHPRC environment variable. (As of PHP 5.2.0)

; 3. A number of predefined registry keys on Windows (As of PHP 5.2.0)

; 4. Current working directory (except CLI)

; 5. The web server's directory (for SAPI modules), or directory of PHP

; (otherwise in Windows)

; 6. The directory from the --with-config-file-path compile time option, or the

; Windows directory (usually C:\windows)

; See the PHP docs for more specific information.

; http://php.net/configuration.file

; The syntax of the file is extremely simple.  Whitespace and lines

; beginning with a semicolon are silently ignored (as you probably guessed).

; Section headers (e.g. [Foo]) are also silently ignored, even though

; they might mean something in the future.

; Directives following the section heading [PATH=/www/mysite] only
```

```
; apply to PHP files in the /www/mysite directory.  Directives

; following the section heading [HOST=www.example.com] only apply to

; PHP files served from www.example.com.  Directives set in these

; special sections cannot be overridden by user-defined INI files or

; at runtime. Currently, [PATH=] and [HOST=] sections only work under

; CGI/FastCGI.

; http://php.net/ini.sections

; Directives are specified using the following syntax:

; directive = value

; Directive names are *case sensitive* - foo=bar is different from FOO=bar.

; Directives are variables used to configure PHP or PHP extensions.

; There is no name validation.  If PHP can't find an expected

; directive because it is not set or is mistyped, a default value will be used.

; The value can be a string, a number, a PHP constant (e.g. E_ALL or M_PI), one

; of the INI constants (On, Off, True, False, Yes, No and None) or an expression

; (e.g. E_ALL & ~E_NOTICE), a quoted string ("bar"), or a reference to a

; previously set variable or directive (e.g. ${foo})

; Expressions in the INI file are limited to bitwise operators and parentheses:

; |  bitwise OR

; ^  bitwise XOR

; &  bitwise AND

; ~  bitwise NOT

; !  boolean NOT

; Boolean flags can be turned on using the values 1, On, True or Yes.

; They can be turned off using the values 0, Off, False or No.

; An empty string can be denoted by simply not writing anything after the equal

; sign, or by using the None keyword:

; foo =         ; sets foo to an empty string

; foo = None    ; sets foo to an empty string

; foo = "None"  ; sets foo to the string 'None'

; If you use constants in your value, and these constants belong to a

; dynamically loaded extension (either a PHP extension or a Zend extension),

; you may only use these constants *after* the line that loads the extension.

;;;;;;;;;;;;;;;;;;;

; About this file ;

;;;;;;;;;;;;;;;;;;;

; PHP comes packaged with two INI files. One that is recommended to be used
```

```
; in production environments and one that is recommended to be used in
; development environments.
; php.ini-production contains settings which hold security, performance and
; best practices at its core. But please be aware, these settings may break
; compatibility with older or less security conscience applications. We
; recommending using the production ini in production and testing environments.
; php.ini-development is very similar to its production variant, except it is
; much more verbose when it comes to errors. We recommend using the
; development version only in development environments, as errors shown to
; application users can inadvertently leak otherwise secure information.
; This is the php.ini-development INI file.
;;;;;;;;;;;;;;;;;;;
; Quick Reference ;
;;;;;;;;;;;;;;;;;;;
; The following are all the settings which are different in either the production
; or development versions of the INIs with respect to PHP's default behavior.
; Please see the actual settings later in the document for more details as to why
; we recommend these changes in PHP's behavior.
; display_errors
;   Default Value: On
;   Development Value: On
;   Production Value: Off
; display_startup_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: Off
; error_reporting
;   Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
;   Development Value: E_ALL
;   Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT
; html_errors
;   Default Value: On
;   Development Value: On
;   Production value: On
; log_errors
;   Default Value: Off
;   Development Value: On
```

```
;   Production Value: On

; max_input_time

;   Default Value: -1 (Unlimited)

;   Development Value: 60 (60 seconds)

;   Production Value: 60 (60 seconds)

; output_buffering

;   Default Value: Off

;   Development Value: 4096

;   Production Value: 4096

; register_argc_argv

;   Default Value: On

;   Development Value: Off

;   Production Value: Off

; request_order

;   Default Value: None

;   Development Value: "GP"

;   Production Value: "GP"

; session.gc_divisor

;   Default Value: 100

;   Development Value: 1000

;   Production Value: 1000

; session.sid_bits_per_character

;   Default Value: 4

;   Development Value: 5

;   Production Value: 5

; short_open_tag

;   Default Value: On

;   Development Value: Off

;   Production Value: Off

; variables_order

;   Default Value: "EGPCS"

;   Development Value: "GPCS"

;   Production Value: "GPCS"

;;;;;;;;;;;;;;;;;;;;

; php.ini Options  ;

;;;;;;;;;;;;;;;;;;;;

; Name for user-defined php.ini (.htaccess) files. Default is ".user.ini"
```

```
;user_ini.filename = ".user.ini"

; To disable this feature set this option to an empty value

;user_ini.filename =

; TTL for user-defined php.ini files (time-to-live) in seconds. Default is 300 seconds (5 minutes)

;user_ini.cache_ttl = 300

;;;;;;;;;;;;;;;;;;;;

; Language Options ;

;;;;;;;;;;;;;;;;;;;;

; Enable the PHP scripting language engine under Apache.

; http://php.net/engine

engine=On

; This directive determines whether or not PHP will recognize code between

; <? and ?> tags as PHP source which should be processed as such. It is

; generally recommended that <?php and ?> should be used and that this feature

; should be disabled, as enabling it may result in issues when generating XML

; documents, however this remains supported for backward compatibility reasons.

; Note that this directive does not control the <?= shorthand tag, which can be

; used regardless of this directive.

; Default Value: On

; Development Value: Off

; Production Value: Off

; http://php.net/short-open-tag

short_open_tag=Off

; The number of significant digits displayed in floating point numbers.

; http://php.net/precision

precision=14

; Output buffering is a mechanism for controlling how much output data

; (excluding headers and cookies) PHP should keep internally before pushing that

; data to the client. If your application's output exceeds this setting, PHP

; will send that data in chunks of roughly the size you specify.

; Turning on this setting and managing its maximum buffer size can yield some

; interesting side-effects depending on your application and web server.

; You may be able to send headers and cookies after you've already sent output

; through print or echo. You also may see performance benefits if your server is

; emitting less packets due to buffered output versus PHP streaming the output

; as it gets it. On production servers, 4096 bytes is a good setting for performance

; reasons.
```

```
; Note: Output buffering can also be controlled via Output Buffering Control
;   functions.
; Possible Values:
;   On = Enabled and buffer is unlimited. (Use with caution)
;   Off = Disabled
;   Integer = Enables the buffer and sets its maximum size in bytes.
; Note: This directive is hardcoded to Off for the CLI SAPI
; Default Value: Off
; Development Value: 4096
; Production Value: 4096
; http://php.net/output-buffering
output_buffering=4096
; You can redirect all of the output of your scripts to a function.  For
; example, if you set output_handler to "mb_output_handler", character
; encoding will be transparently converted to the specified encoding.
; Setting any output handler automatically turns on output buffering.
; Note: People who wrote portable scripts should not depend on this ini
;   directive. Instead, explicitly set the output handler using ob_start().
;   Using this ini directive may cause problems unless you know what script
;   is doing.
; Note: You cannot use both "mb_output_handler" with "ob_iconv_handler"
;   and you cannot use both "ob_gzhandler" and "zlib.output_compression".
; Note: output_handler must be empty if this is set 'On' !!!!
;   Instead you must use zlib.output_handler.
; http://php.net/output-handler
;output_handler =
; URL rewriter function rewrites URL on the fly by using
; output buffer. You can set target tags by this configuration.
; "form" tag is special tag. It will add hidden input tag to pass values.
; Refer to session.trans_sid_tags for usage.
; Default Value: "form="
; Development Value: "form="
; Production Value: "form="
;url_rewriter.tags
; URL rewriter will not rewrite absolute URL nor form by default. To enable
; absolute URL rewrite, allowed hosts must be defined at RUNTIME.
; Refer to session.trans_sid_hosts for more details.
```

```
; Default Value: ""

; Development Value: ""

; Production Value: ""

;url_rewriter.hosts

; Transparent output compression using the zlib library

; Valid values for this option are 'off', 'on', or a specific buffer size

; to be used for compression (default is 4KB)

; Note: Resulting chunk size may vary due to nature of compression. PHP

;   outputs chunks that are few hundreds bytes each as a result of

;   compression. If you prefer a larger chunk size for better

;   performance, enable output_buffering in addition.

; Note: You need to use zlib.output_handler instead of the standard

;   output_handler, or otherwise the output will be corrupted.

; http://php.net/zlib.output-compression

zlib.output_compression=Off

; http://php.net/zlib.output-compression-level

;zlib.output_compression_level = -1

; You cannot specify additional output handlers if zlib.output_compression

; is activated here. This setting does the same as output_handler but in

; a different order.

; http://php.net/zlib.output-handler

;zlib.output_handler =

; Implicit flush tells PHP to tell the output layer to flush itself

; automatically after every output block.  This is equivalent to calling the

; PHP function flush() after each and every call to print() or echo() and each

; and every HTML block.  Turning this option on has serious performance

; implications and is generally recommended for debugging purposes only.

; http://php.net/implicit-flush

; Note: This directive is hardcoded to On for the CLI SAPI

implicit_flush=Off

; The unserialize callback function will be called (with the undefined class'

; name as parameter), if the unserializer finds an undefined class

; which should be instantiated. A warning appears if the specified function is

; not defined, or if the function doesn't include/implement the missing class.

; So only set this entry, if you really want to implement such a

; callback-function.

unserialize_callback_func=
```

```
; When floats & doubles are serialized, store serialize_precision significant
; digits after the floating point. The default value ensures that when floats
; are decoded with unserialize, the data will remain the same.
; The value is also used for json_encode when encoding double values.
; If -1 is used, then dtoa mode 0 is used which automatically select the best
; precision.
serialize_precision=-1
; open_basedir, if set, limits all file operations to the defined directory
; and below.  This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file.
; Note: disables the realpath cache
; http://php.net/open-basedir
;open_basedir =
; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names.
; http://php.net/disable-functions
disable_functions=
; This directive allows you to disable certain classes for security reasons.
; It receives a comma-delimited list of class names.
; http://php.net/disable-classes
disable_classes=
; Colors for Syntax Highlighting mode.  Anything that's acceptable in
; <span style="color: ???????"> would work.
; http://php.net/syntax-highlighting
;highlight.string  = #DD0000
;highlight.comment = #FF9900
;highlight.keyword = #007700
;highlight.default = #0000BB
;highlight.html    = #000000
; If enabled, the request will be allowed to complete even if the user aborts
; the request. Consider enabling it if executing long requests, which may end up
; being interrupted by the user or a browser timing out. PHP's default behavior
; is to disable this feature.
; http://php.net/ignore-user-abort
;ignore_user_abort = On
; Determines the size of the realpath cache to be used by PHP. This value should
; be increased on systems where PHP opens many files to reflect the quantity of
```

```
; the file operations performed.

; Note: if open_basedir is set, the cache is disabled

; http://php.net/realpath-cache-size

;realpath_cache_size = 4096k

; Duration of time, in seconds for which to cache realpath information for a given

; file or directory. For systems with rarely changing files, consider increasing this

; value.

; http://php.net/realpath-cache-ttl

;realpath_cache_ttl = 120

; Enables or disables the circular reference collector.

; http://php.net/zend.enable-gc

zend.enable_gc=On

; If enabled, scripts may be written in encodings that are incompatible with

; the scanner.  CP936, Big5, CP949 and Shift_JIS are the examples of such

; encodings.  To use this feature, mbstring extension must be enabled.

; Default: Off

;zend.multibyte = Off

; Allows to set the default encoding for the scripts.  This value will be used

; unless "declare(encoding=...)" directive appears at the top of the script.

; Only affects if zend.multibyte is set.

; Default: ""

;zend.script_encoding =

;;;;;;;;;;;;;;;;;;

; Miscellaneous ;

;;;;;;;;;;;;;;;;;;

; Decides whether PHP may expose the fact that it is installed on the server

; (e.g. by adding its signature to the Web server header).  It is no security

; threat in any way, but it makes it possible to determine whether you use PHP

; on your server or not.

; http://php.net/expose-php

expose_php=On

;;;;;;;;;;;;;;;;;;;

; Resource Limits ;

;;;;;;;;;;;;;;;;;;;

; Maximum execution time of each script, in seconds

; http://php.net/max-execution-time

; Note: This directive is hardcoded to 0 for the CLI SAPI
```

```
max_execution_time=30

; Maximum amount of time each script may spend parsing request data. It's a good

; idea to limit this time on productions servers in order to eliminate unexpectedly

; long running scripts.

; Note: This directive is hardcoded to -1 for the CLI SAPI

; Default Value: -1 (Unlimited)

; Development Value: 60 (60 seconds)

; Production Value: 60 (60 seconds)

; http://php.net/max-input-time

max_input_time=60

; Maximum input variable nesting level

; http://php.net/max-input-nesting-level

;max_input_nesting_level = 64

; How many GET/POST/COOKIE input variables may be accepted

;max_input_vars = 1000

; Maximum amount of memory a script may consume (128MB)

; http://php.net/memory-limit

memory_limit=128M

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Error handling and logging ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This directive informs PHP of which errors, warnings and notices you would like

; it to take action for. The recommended way of setting values for this

; directive is through the use of the error level constants and bitwise

; operators. The error level constants are below here for convenience as well as

; some common settings and their meanings.

; By default, PHP is set to take action on all errors, notices and warnings EXCEPT

; those related to E_NOTICE and E_STRICT, which together cover best practices and

; recommended coding standards in PHP. For performance reasons, this is the

; recommend error reporting setting. Your production server shouldn't be wasting

; resources complaining about best practices and coding standards. That's what

; development servers and development settings are for.

; Note: The php.ini-development file has this setting as E_ALL. This

; means it pretty much reports everything which is exactly what you want during

; development and early testing.

;

; Error Level Constants:
```

```
; E_ALL             - All errors and warnings (includes E_STRICT as of PHP 5.4.0)
; E_ERROR           - fatal run-time errors
; E_RECOVERABLE_ERROR  - almost fatal run-time errors
; E_WARNING         - run-time warnings (non-fatal errors)
; E_PARSE           - compile-time parse errors
; E_NOTICE          - run-time notices (these are warnings which often result
;                     from a bug in your code, but it's possible that it was
;                     intentional (e.g., using an uninitialized variable and
;                     relying on the fact it is automatically initialized to an
;                     empty string)
; E_STRICT          - run-time notices, enable to have PHP suggest changes
;                     to your code which will ensure the best interoperability
;                     and forward compatibility of your code
; E_CORE_ERROR      - fatal errors that occur during PHP's initial startup
; E_CORE_WARNING    - warnings (non-fatal errors) that occur during PHP's
;                     initial startup
; E_COMPILE_ERROR   - fatal compile-time errors
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
; E_USER_ERROR      - user-generated error message
; E_USER_WARNING    - user-generated warning message
; E_USER_NOTICE     - user-generated notice message
; E_DEPRECATED      - warn about code that will not work in future versions
;                     of PHP
; E_USER_DEPRECATED - user-generated deprecation warnings
;
; Common Values:
;   E_ALL (Show all errors, warnings and notices including coding standards.)
;   E_ALL & ~E_NOTICE  (Show all errors, except for notices)
;   E_ALL & ~E_NOTICE & ~E_STRICT  (Show all errors, except for notices and coding standards warnings.)
;   E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR  (Show only errors)
; Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
; Development Value: E_ALL
; Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT
; http://php.net/error-reporting
error_reporting=E_ALL & ~E_DEPRECATED & ~E_STRICT
; This directive controls whether or not and where PHP will output errors,
; notices and warnings too. Error output is very useful during development, but
```

```
; it could be very dangerous in production environments. Depending on the code
; which is triggering the error, sensitive information could potentially leak
; out of your application such as database usernames and passwords or worse.
; For production environments, we recommend logging errors rather than
; sending them to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
; http://php.net/display-errors
display_errors=On
; The display of errors which occur during PHP's startup sequence are handled
; separately from display_errors. PHP's default behavior is to suppress those
; errors from clients. Turning the display of startup errors on can be useful in
; debugging configuration problems. We strongly recommend you
; set this to 'off' for production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off
; http://php.net/display-startup-errors
display_startup_errors=On
; Besides displaying errors, PHP can also log errors to locations such as a
; server-specific log, STDERR, or a location specified by the error_log
; directive found below. While errors should not be displayed on productions
; servers they should still be monitored and logging is a great way to do that.
; Default Value: Off
; Development Value: On
; Production Value: On
; http://php.net/log-errors
log_errors=On
; Set maximum length of log_errors. In error_log information about the source is
; added. The default is 1024 and 0 allows to not apply any maximum length at all.
; http://php.net/log-errors-max-len
log_errors_max_len=1024
```

```
; Do not log repeated messages. Repeated errors must occur in same file on same
; line unless ignore_repeated_source is set true.
; http://php.net/ignore-repeated-errors
ignore_repeated_errors=Off
; Ignore source of message when ignoring repeated messages. When this setting
; is On you will not log errors with repeated messages from different files or
; source lines.
; http://php.net/ignore-repeated-source
ignore_repeated_source=Off
; If this parameter is set to Off, then memory leaks will not be shown (on
; stdout or in the log). This has only effect in a debug compile, and if
; error reporting includes E_WARNING in the allowed list
; http://php.net/report-memleaks
report_memleaks=On
; This setting is on by default.
;report_zend_debug = 0
; Store the last error/warning message in $php_errormsg (boolean).
; This directive is DEPRECATED.
; Default Value: Off
; Development Value: Off
; Production Value: Off
; http://php.net/track-errors
;track_errors = Off
; Turn off normal error reporting and emit XML-RPC error XML
; http://php.net/xmlrpc-errors
;xmlrpc_errors = 0
; An XML-RPC faultCode
;xmlrpc_error_number = 0
; When PHP displays or logs an error, it has the capability of formatting the
; error message as HTML for easier reading. This directive controls whether
; the error message is formatted as HTML or not.
; Note: This directive is hardcoded to Off for the CLI SAPI
; Default Value: On
; Development Value: On
; Production value: On
; http://php.net/html-errors
html_errors=On
```

```
; If html_errors is set to On *and* docref_root is not empty, then PHP
; produces clickable error messages that direct to a page describing the error
; or function causing the error in detail.
; You can download a copy of the PHP manual from http://php.net/docs
; and change docref_root to the base URL of your local copy including the
; leading '/'. You must also specify the file extension being used including
; the dot. PHP's default behavior is to leave these settings empty, in which
; case no links to documentation are generated.
; Note: Never use this feature for production boxes.
; http://php.net/docref-root
; Examples
;docref_root = "/phpmanual/"
; http://php.net/docref-ext
;docref_ext = .html
; String to output before an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-prepend-string
; Example:
;error_prepend_string = "<span style='color: #ff0000'>"
; String to output after an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-append-string
; Example:
;error_append_string = "</span>"
; Log errors to specified file. PHP's default behavior is to leave this value
; empty.
; http://php.net/error-log
; Example:
;error_log = php_errors.log
; Log errors to syslog (Event Log on Windows).
;error_log = syslog
; The syslog ident is a string which is prepended to every message logged
; to syslog. Only used when error_log is set to syslog.
;syslog.ident = php
; The syslog facility is used to specify what type of program is logging
; the message. Only used when error_log is set to syslog.
;syslog.facility = user
```

```
; Set this to disable filtering control characters (the default).

; Some loggers only accept NVT-ASCII, others accept anything that's not

; control characters. If your logger accepts everything, then no filtering

; is needed at all.

; Allowed values are:

;   ascii (all printable ASCII characters and NL)

;   no-ctrl (all characters except control characters)

;   all (all characters)

; http://php.net/syslog.filter

;syslog.filter = ascii

;windows.show_crt_warning

; Default value: 0

; Development value: 0

; Production value: 0

;;;;;;;;;;;;;;;;;

; Data Handling ;

;;;;;;;;;;;;;;;;;

; The separator used in PHP generated URLs to separate arguments.

; PHP's default setting is "&".

; http://php.net/arg-separator.output

; Example:

;arg_separator.output = "&amp;"

; List of separator(s) used by PHP to parse input URLs into variables.

; PHP's default setting is "&".

; NOTE: Every character in this directive is considered as separator!

; http://php.net/arg-separator.input

; Example:

;arg_separator.input = ";&"

; This directive determines which super global arrays are registered when PHP

; starts up. G,P,C,E & S are abbreviations for the following respective super

; globals: GET, POST, COOKIE, ENV and SERVER. There is a performance penalty

; paid for the registration of these arrays and because ENV is not as commonly

; used as the others, ENV is not recommended on productions servers. You

; can still get access to the environment variables through getenv() should you

; need to.

; Default Value: "EGPCS"

; Development Value: "GPCS"
```

```
; Production Value: "GPCS";

; http://php.net/variables-order

variables_order="GPCS"

; This directive determines which super global data (G,P & C) should be

; registered into the super global array REQUEST. If so, it also determines

; the order in which that data is registered. The values for this directive

; are specified in the same manner as the variables_order directive,

; EXCEPT one. Leaving this value empty will cause PHP to use the value set

; in the variables_order directive. It does not mean it will leave the super

; globals array REQUEST empty.

; Default Value: None

; Development Value: "GP"

; Production Value: "GP"

; http://php.net/request-order

request_order="GP"

; This directive determines whether PHP registers $argv & $argc each time it

; runs. $argv contains an array of all the arguments passed to PHP when a script

; is invoked. $argc contains an integer representing the number of arguments

; that were passed when the script was invoked. These arrays are extremely

; useful when running scripts from the command line. When this directive is

; enabled, registering these variables consumes CPU cycles and memory each time

; a script is executed. For performance reasons, this feature should be disabled

; on production servers.

; Note: This directive is hardcoded to On for the CLI SAPI

; Default Value: On

; Development Value: Off

; Production Value: Off

; http://php.net/register-argc-argv

register_argc_argv=Off

; When enabled, the ENV, REQUEST and SERVER variables are created when they're

; first used (Just In Time) instead of when the script starts. If these

; variables are not used within a script, having this directive on will result

; in a performance gain. The PHP directive register_argc_argv must be disabled

; for this directive to have any affect.

; http://php.net/auto-globals-jit

auto_globals_jit=On

; Whether PHP will read the POST data.
```

```
; This option is enabled by default.
; Most likely, you won't want to disable this option globally. It causes $_POST
; and $_FILES to always be empty; the only way you will be able to read the
; POST data will be through the php://input stream wrapper. This can be useful
; to proxy requests or to process the POST data in a memory efficient fashion.
; http://php.net/enable-post-data-reading
;enable_post_data_reading = Off
; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; http://php.net/post-max-size
post_max_size=8M
; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file=
; Automatically add files after PHP document.
; http://php.net/auto-append-file
auto_append_file=
; By default, PHP will output a media type using the Content-Type header. To
; disable this, simply set it to be empty.
;
; PHP's built-in default media type is set to text/html.
; http://php.net/default-mimetype
default_mimetype="text/html"
; PHP's default character set is set to UTF-8.
; http://php.net/default-charset
default_charset="UTF-8"
; PHP internal character encoding is set to empty.
; If empty, default_charset is used.
; http://php.net/internal-encoding
;internal_encoding =
; PHP input character encoding is set to empty.
; If empty, default_charset is used.
; http://php.net/input-encoding
;input_encoding =
; PHP output character encoding is set to empty.
; If empty, default_charset is used.
```

```
; See also output_buffer.
; http://php.net/output-encoding
;output_encoding =
;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;
; UNIX: "/path1:/path2"
include_path=C:\xampp\php\PEAR
;
; Windows: "\path1;\path2"
;include_path = ".;c:\php\includes"
;
; PHP's default setting for include_path is ".;/path/to/php/pear"
; http://php.net/include-path
; The root of the PHP pages, used only if nonempty.
; if PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root
; if you are running php as a CGI under any web server (other than IIS)
; see documentation for security issues.  The alternate is to use the
; cgi.force_redirect configuration below
; http://php.net/doc-root
doc_root=
; The directory under which PHP opens the script using /~username used only
; if nonempty.
; http://php.net/user-dir
user_dir=
; Directory in which the loadable extensions (modules) reside.
; http://php.net/extension-dir
;extension_dir = "./"
; On windows:
extension_dir="C:\xampp\php\ext"
; Directory where the temporary files should be placed.
; Defaults to the system default (see sys_get_temp_dir)
;sys_temp_dir = "/tmp"
; Whether or not to enable the dl() function.  The dl() function does NOT work
; properly in multithreaded servers, such as IIS or Zeus, and is automatically
; disabled on them.
; http://php.net/enable-dl
```

```
enable_dl=Off

; cgi.force_redirect is necessary to provide security running PHP as a CGI under

; most web servers.  Left undefined, PHP turns this on by default.  You can

; turn it off here AT YOUR OWN RISK

; **You CAN safely turn this off for IIS, in fact, you MUST.**

; http://php.net/cgi.force-redirect

;cgi.force_redirect = 1

; if cgi.nph is enabled it will force cgi to always sent Status: 200 with

; every request. PHP's default behavior is to disable this feature.

;cgi.nph = 1

; if cgi.force_redirect is turned on, and you are not running under Apache or Netscape

; (iPlanet) web servers, you MAY need to set an environment variable name that PHP

; will look for to know it is OK to continue execution.  Setting this variable MAY

; cause security issues, KNOW WHAT YOU ARE DOING FIRST.

; http://php.net/cgi.redirect-status-env

;cgi.redirect_status_env =

; cgi.fix_pathinfo provides *real* PATH_INFO/PATH_TRANSLATED support for CGI.  PHP's

; previous behaviour was to set PATH_TRANSLATED to SCRIPT_FILENAME, and to not grok

; what PATH_INFO is.  For more information on PATH_INFO, see the cgi specs.  Setting

; this to 1 will cause PHP CGI to fix its paths to conform to the spec.  A setting

; of zero causes PHP to behave as before.  Default is 1.  You should fix your scripts

; to use SCRIPT_FILENAME rather than PATH_TRANSLATED.

; http://php.net/cgi.fix-pathinfo

;cgi.fix_pathinfo=1

; if cgi.discard_path is enabled, the PHP CGI binary can safely be placed outside

; of the web tree and people will not be able to circumvent .htaccess security.

;cgi.discard_path=1

; FastCGI under IIS supports the ability to impersonate

; security tokens of the calling client.  This allows IIS to define the

; security context that the request runs under.  mod_fastcgi under Apache

; does not currently support this feature (03/17/2002)

; Set to 1 if running under IIS.  Default is zero.

; http://php.net/fastcgi.impersonate

;fastcgi.impersonate = 1

; Disable logging through FastCGI connection. PHP's default behavior is to enable

; this feature.

;fastcgi.logging = 0
```

```
; cgi.rfc2616_headers configuration option tells PHP what type of headers to
; use when sending HTTP response code. If set to 0, PHP sends Status: header that
; is supported by Apache. When this option is set to 1, PHP will send
; RFC2616 compliant header.
; Default is zero.
; http://php.net/cgi.rfc2616-headers
;cgi.rfc2616_headers = 0
; cgi.check_shebang_line controls whether CGI PHP checks for line starting with #!
; (shebang) at the top of the running script. This line might be needed if the
; script support running both as stand-alone script and via PHP CGI<. PHP in CGI
; mode skips this line and ignores its content if this directive is turned on.
; http://php.net/cgi.check-shebang-line
;cgi.check_shebang_line=1
;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;
; Whether to allow HTTP file uploads.
; http://php.net/file-uploads
file_uploads=On
; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; http://php.net/upload-tmp-dir
upload_tmp_dir="C:\xampp\tmp"
; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize=2M
; Maximum number of files that can be uploaded via a single request
max_file_uploads=20
;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;
; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-fopen
allow_url_fopen=On
; Whether to allow include/require to open URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-include
allow_url_include=Off
```

```
; Define the anonymous ftp password (your email address). PHP's default setting
; for this is empty.
; http://php.net/from
;from="john@doe.com"
; Define the User-Agent string. PHP's default setting for this is empty.
; http://php.net/user-agent
;user_agent="PHP"
; Default timeout for socket based streams (seconds)
; http://php.net/default-socket-timeout
default_socket_timeout=60
; If your scripts have to deal with files from Macintosh systems,
; or you are running on a Mac and need to deal with files from
; unix or win32 systems, setting this flag will cause PHP to
; automatically detect the EOL character in those files so that
; fgets() and file() will work regardless of the source of the file.
; http://php.net/auto-detect-line-endings
;auto_detect_line_endings = Off
;;;;;;;;;;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
;   extension=modulename
;
; For example:
;
;   extension=mysqli
;
; When the extension library to load is not located in the default extension
; directory, You may specify an absolute path to the library file:
;
;   extension=/path/to/extension/mysqli.so
;
; Note : The syntax used in previous PHP versions ('extension=<ext>.so' and
; 'extension='php_<ext>.dll') is supported for legacy reasons and may be
; deprecated in a future PHP major version. So, when it is possible, please
```

```
; move to the new ('extension=<ext>) syntax.
;
; Notes for Windows environments :
;
; - Many DLL files are located in the extensions/ (PHP 4) or ext/ (PHP 5+)
;   extension folders as well as the separate PECL DLL download (PHP 5+).
;   Be sure to appropriately set the extension_dir directive.
;
extension=bz2
extension=curl
extension=fileinfo
extension=gd2
extension=gettext
;extension=gmp
;extension=intl
;extension=imap
;extension=interbase
;extension=ldap
extension=mbstring
extension=exif      ; Must be after mbstring as it depends on it
extension=mysqli
;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
;extension=odbc
;extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
;extension=pdo_pgsql
extension=pdo_sqlite
;extension=pgsql
;extension=shmop
; The MIBS data available in the PHP distribution must be installed.
; See http://www.php.net/manual/en/snmp.installation.php
;extension=snmp
;extension=soap
;extension=sockets
```

```
;extension=sodium

;extension=sqlite3

;extension=tidy

;extension=xmlrpc

;extension=xsl

;;;;;;;;;;;;;;;;;;;

; Module Settings ;

;;;;;;;;;;;;;;;;;;;

asp_tags=Off

display_startup_errors=On

track_errors=Off

y2k_compliance=On

allow_call_time_pass_reference=Off

safe_mode=Off

safe_mode_gid=Off

safe_mode_allowed_env_vars=PHP_

safe_mode_protected_env_vars=LD_LIBRARY_PATH

error_log="C:\xampp\php\logs\php_error_log"

register_globals=Off

register_long_arrays=Off

magic_quotes_gpc=Off

magic_quotes_runtime=Off

magic_quotes_sybase=Off

extension=php_openssl.dll

extension=php_ftp.dll

[CLI Server]

; Whether the CLI web server uses ANSI color coding in its terminal output.

cli_server.color=On

[Date]

; Defines the default timezone used by the date functions

; http://php.net/date.timezone

;date.timezone =

; http://php.net/date.default-latitude

;date.default_latitude = 31.7667

; http://php.net/date.default-longitude

;date.default_longitude = 35.2333

; http://php.net/date.sunrise-zenith
```

```
;date.sunrise_zenith = 90.583333

; http://php.net/date.sunset-zenith

;date.sunset_zenith = 90.583333

[filter]

; http://php.net/filter.default

;filter.default = unsafe_raw

; http://php.net/filter.default-flags

;filter.default_flags =

[iconv]

; Use of this INI entry is deprecated, use global input_encoding instead.

; If empty, default_charset or input_encoding or iconv.input_encoding is used.

; The precedence is: default_charset < input_encoding < iconv.input_encoding

;iconv.input_encoding =

; Use of this INI entry is deprecated, use global internal_encoding instead.

; If empty, default_charset or internal_encoding or iconv.internal_encoding is used.

; The precedence is: default_charset < internal_encoding < iconv.internal_encoding

;iconv.internal_encoding =

; Use of this INI entry is deprecated, use global output_encoding instead.

; If empty, default_charset or output_encoding or iconv.output_encoding is used.

; The precedence is: default_charset < output_encoding < iconv.output_encoding

; To use an output encoding conversion, iconv's output handler must be set

; otherwise output encoding conversion cannot be performed.

;iconv.output_encoding =

[imap]

; rsh/ssh logins are disabled by default. Use this INI entry if you want to

; enable them. Note that the IMAP library does not filter mailbox names before

; passing them to rsh/ssh command, thus passing untrusted data to this function

; with rsh/ssh enabled is insecure.

;imap.enable_insecure_rsh=0

[intl]

;intl.default_locale =

; This directive allows you to produce PHP errors when some error

; happens within intl functions. The value is the level of the error produced.

; Default is 0, which does not produce any errors.

;intl.error_level = E_WARNING

;intl.use_exceptions = 0

[sqlite3]
```

```
; Directory pointing to SQLite3 extensions
; http://php.net/sqlite3.extension-dir
;sqlite3.extension_dir =
; SQLite defensive mode flag (only available from SQLite 3.26+)
; When the defensive flag is enabled, language features that allow ordinary
; SQL to deliberately corrupt the database file are disabled. This forbids
; writing directly to the schema, shadow tables (eg. FTS data tables), or
; the sqlite_dbpage virtual table.
; https://www.sqlite.org/c3ref/c_dbconfig_defensive.html
; (for older SQLite versions, this flag has no use)
;sqlite3.defensive = 1
[Pcre]
; PCRE library backtracking limit.
; http://php.net/pcre.backtrack-limit
;pcre.backtrack_limit=100000
; PCRE library recursion limit.
; Please note that if you set this value to a high number you may consume all
; the available process stack and eventually crash PHP (due to reaching the
; stack size limit imposed by the Operating System).
; http://php.net/pcre.recursion-limit
;pcre.recursion_limit=100000
; Enables or disables JIT compilation of patterns. This requires the PCRE
; library to be compiled with JIT support.
;pcre.jit=1
[Pdo]
pdo_mysql.default_socket="MySQL"
; Whether to pool ODBC connections. Can be one of "strict", "relaxed" or "off"
; http://php.net/pdo-odbc.connection-pooling
;pdo_odbc.connection_pooling=strict
;pdo_odbc.db2_instance_name
[Pdo_mysql]
; Default socket name for local MySQL connects.  If empty, uses the built-in
; MySQL defaults.
pdo_mysql.default_socket=
[Phar]
; http://php.net/phar.readonly
;phar.readonly = On
```

```
; http://php.net/phar.require-hash

;phar.require_hash = On

;phar.cache_list =

[mail function]

; For Win32 only.

; http://php.net/smtp

SMTP=localhost

; http://php.net/smtp-port

smtp_port=25

; For Win32 only.

; http://php.net/sendmail-from

;sendmail_from = me@example.com

; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").

; http://php.net/sendmail-path

;sendmail_path =

; Force the addition of the specified parameters to be passed as extra parameters

; to the sendmail binary. These parameters will always replace the value of

; the 5th parameter to mail().

;mail.force_extra_parameters =

; Add X-PHP-Originating-Script: that will include uid of the script followed by the filename

mail.add_x_header=Off

; The path to a log file that will log all mail() calls. Log entries include

; the full path of the script, line number, To address and headers.

;mail.log =

; Log mail to syslog (Event Log on Windows).

;mail.log = syslog

[ODBC]

; http://php.net/odbc.default-db

;odbc.default_db    =  Not yet implemented

; http://php.net/odbc.default-user

;odbc.default_user  =  Not yet implemented

; http://php.net/odbc.default-pw

;odbc.default_pw    =  Not yet implemented

; Controls the ODBC cursor model.

; Default: SQL_CURSOR_STATIC (default).

;odbc.default_cursortype

; Allow or prevent persistent links.
```

```
; http://php.net/odbc.allow-persistent

odbc.allow_persistent=On

; Check that a connection is still valid before reuse.

; http://php.net/odbc.check-persistent

odbc.check_persistent=On

; Maximum number of persistent links.  -1 means no limit.

; http://php.net/odbc.max-persistent

odbc.max_persistent=-1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.

; http://php.net/odbc.max-links

odbc.max_links=-1

; Handling of LONG fields.  Returns number of bytes to variables.  0 means

; passthru.

; http://php.net/odbc.defaultlrl

odbc.defaultlrl=4096

; Handling of binary data.  0 means passthru, 1 return as is, 2 convert to char.

; See the documentation on odbc_binmode and odbc_longreadlen for an explanation

; of odbc.defaultlrl and odbc.defaultbinmode

; http://php.net/odbc.defaultbinmode

odbc.defaultbinmode=1

[Interbase]

; Allow or prevent persistent links.

ibase.allow_persistent=1

; Maximum number of persistent links.  -1 means no limit.

ibase.max_persistent=-1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.

ibase.max_links=-1

; Default database name for ibase_connect().

;ibase.default_db =

; Default username for ibase_connect().

;ibase.default_user =

; Default password for ibase_connect().

;ibase.default_password =

; Default charset for ibase_connect().

;ibase.default_charset =

; Default timestamp format.

ibase.timestampformat="%Y-%m-%d %H:%M:%S"
```

```
; Default date format.

ibase.dateformat="%Y-%m-%d"

; Default time format.

ibase.timeformat="%H:%M:%S"

[MySQLi]

; Maximum number of persistent links.  -1 means no limit.

; http://php.net/mysqli.max-persistent

mysqli.max_persistent=-1

; Allow accessing, from PHP's perspective, local files with LOAD DATA statements

; http://php.net/mysqli.allow_local_infile

;mysqli.allow_local_infile = On

; Allow or prevent persistent links.

; http://php.net/mysqli.allow-persistent

mysqli.allow_persistent=On

; Maximum number of links.  -1 means no limit.

; http://php.net/mysqli.max-links

mysqli.max_links=-1

; Default port number for mysqli_connect().  If unset, mysqli_connect() will use

; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the

; compile-time value defined MYSQL_PORT (in that order).  Win32 will only look

; at MYSQL_PORT.

; http://php.net/mysqli.default-port

mysqli.default_port=3306

; Default socket name for local MySQL connects.  If empty, uses the built-in

; MySQL defaults.

; http://php.net/mysqli.default-socket

mysqli.default_socket=

; Default host for mysql_connect() (doesn't apply in safe mode).

; http://php.net/mysqli.default-host

mysqli.default_host=

; Default user for mysql_connect() (doesn't apply in safe mode).

; http://php.net/mysqli.default-user

mysqli.default_user=

; Default password for mysqli_connect() (doesn't apply in safe mode).

; Note that this is generally a *bad* idea to store passwords in this file.

; *Any* user with PHP access can run 'echo get_cfg_var("mysqli.default_pw")

; and reveal this password!  And of course, any users with read access to this
```

```
; file will be able to reveal the password as well.

; http://php.net/mysqli.default-pw

mysqli.default_pw=

; Allow or prevent reconnect

mysqli.reconnect=Off

[mysqlnd]

; Enable / Disable collection of general statistics by mysqlnd which can be

; used to tune and monitor MySQL operations.

mysqlnd.collect_statistics=On

; Enable / Disable collection of memory usage statistics by mysqlnd which can be

; used to tune and monitor MySQL operations.

mysqlnd.collect_memory_statistics=On

; Records communication from all extensions using mysqlnd to the specified log

; file.

; http://php.net/mysqlnd.debug

;mysqlnd.debug =

; Defines which queries will be logged.

;mysqlnd.log_mask = 0

; Default size of the mysqlnd memory pool, which is used by result sets.

;mysqlnd.mempool_default_size = 16000

; Size of a pre-allocated buffer used when sending commands to MySQL in bytes.

;mysqlnd.net_cmd_buffer_size = 2048

; Size of a pre-allocated buffer used for reading data sent by the server in

; bytes.

;mysqlnd.net_read_buffer_size = 32768

; Timeout for network requests in seconds.

;mysqlnd.net_read_timeout = 31536000

; SHA-256 Authentication Plugin related. File with the MySQL server public RSA

; key.

;mysqlnd.sha256_server_public_key =

[OCI8]

; Connection: Enables privileged connections using external

; credentials (OCI_SYSOPER, OCI_SYSDBA)

; http://php.net/oci8.privileged-connect

;oci8.privileged_connect = Off

; Connection: The maximum number of persistent OCI8 connections per

; process. Using -1 means no limit.
```

```
; http://php.net/oci8.max-persistent
;oci8.max_persistent = -1
; Connection: The maximum number of seconds a process is allowed to
; maintain an idle persistent connection. Using -1 means idle
; persistent connections will be maintained forever.
; http://php.net/oci8.persistent-timeout
;oci8.persistent_timeout = -1
; Connection: The number of seconds that must pass before issuing a
; ping during oci_pconnect() to check the connection validity. When
; set to 0, each oci_pconnect() will cause a ping. Using -1 disables
; pings completely.
; http://php.net/oci8.ping-interval
;oci8.ping_interval = 60
; Connection: Set this to a user chosen connection class to be used
; for all pooled server requests with Oracle 11g Database Resident
; Connection Pooling (DRCP).  To use DRCP, this value should be set to
; the same string for all web servers running the same application,
; the database pool must be configured, and the connection string must
; specify to use a pooled server.
;oci8.connection_class =
; High Availability: Using On lets PHP receive Fast Application
; Notification (FAN) events generated when a database node fails. The
; database must also be configured to post FAN events.
;oci8.events = Off
; Tuning: This option enables statement caching, and specifies how
; many statements to cache. Using 0 disables statement caching.
; http://php.net/oci8.statement-cache-size
;oci8.statement_cache_size = 20
; Tuning: Enables statement prefetching and sets the default number of
; rows that will be fetched automatically after statement execution.
; http://php.net/oci8.default-prefetch
;oci8.default_prefetch = 100
; Compatibility. Using On means oci_close() will not close
; oci_connect() and oci_new_connect() connections.
; http://php.net/oci8.old-oci-close-semantics
;oci8.old_oci_close_semantics = Off
[PostgreSQL]
```

```
; Allow or prevent persistent links.

; http://php.net/pgsql.allow-persistent

pgsql.allow_persistent=On

; Detect broken persistent links always with pg_pconnect().

; Auto reset feature requires a little overheads.

; http://php.net/pgsql.auto-reset-persistent

pgsql.auto_reset_persistent=Off

; Maximum number of persistent links.  -1 means no limit.

; http://php.net/pgsql.max-persistent

pgsql.max_persistent=-1

; Maximum number of links (persistent+non persistent).  -1 means no limit.

; http://php.net/pgsql.max-links

pgsql.max_links=-1

; Ignore PostgreSQL backends Notice message or not.

; Notice message logging require a little overheads.

; http://php.net/pgsql.ignore-notice

pgsql.ignore_notice=0

; Log PostgreSQL backends Notice message or not.

; Unless pgsql.ignore_notice=0, module cannot log notice message.

; http://php.net/pgsql.log-notice

pgsql.log_notice=0

[bcmath]

; Number of decimal digits for all bcmath functions.

; http://php.net/bcmath.scale

bcmath.scale=0

[browscap]

; http://php.net/browscap

browscap="C:\xampp\php\extras\browscap.ini"

[Session]

; Handler used to store/retrieve data.

; http://php.net/session.save-handler

session.save_handler=files

; Argument passed to save_handler.  In the case of files, this is the path

; where data files are stored. Note: Windows users have to change this

; variable in order to use PHP's session functions.

;

; The path can be defined as:
```

```
;
;     session.save_path = "N;/path"
;
; where N is an integer.  Instead of storing all the session files in
; /path, what this will do is use subdirectories N-levels deep, and
; store the session data in those directories.  This is useful if
; your OS has problems with many files in one directory, and is
; a more efficient layout for servers that handle many sessions.
;
; NOTE 1: PHP will not create this directory structure automatically.
;         You can use the script in the ext/session dir for that purpose.
; NOTE 2: See the section on garbage collection below if you choose to
;         use subdirectories for session storage
;
; The file storage module creates files using mode 600 by default.
; You can change that by using
;
;     session.save_path = "N;MODE;/path"
;
; where MODE is the octal representation of the mode. Note that this
; does not overwrite the process's umask.
; http://php.net/session.save-path
session.save_path="C:\xampp\tmp"
; Whether to use strict session mode.
; Strict session mode does not accept an uninitialized session ID, and
; regenerates the session ID if the browser sends an uninitialized session ID.
; Strict mode protects applications from session fixation via a session adoption
; vulnerability. It is disabled by default for maximum compatibility, but
; enabling it is encouraged.
; https://wiki.php.net/rfc/strict_sessions
session.use_strict_mode=0
; Whether to use cookies.
; http://php.net/session.use-cookies
session.use_cookies=1
; http://php.net/session.cookie-secure
;session.cookie_secure =
; This option forces PHP to fetch and use a cookie for storing and maintaining
```

```
; the session id. We encourage this operation as it's very helpful in combating
; session hijacking when not specifying and managing your own session id. It is
; not the be-all and end-all of session hijacking defense, but it's a good start.
; http://php.net/session.use-only-cookies
session.use_only_cookies=1
; Name of the session (used as cookie name).
; http://php.net/session.name
session.name=PHPSESSID
; Initialize session on request startup.
; http://php.net/session.auto-start
session.auto_start=0
; Lifetime in seconds of cookie or, if 0, until browser is restarted.
; http://php.net/session.cookie-lifetime
session.cookie_lifetime=0
; The path for which the cookie is valid.
; http://php.net/session.cookie-path
session.cookie_path=/
; The domain for which the cookie is valid.
; http://php.net/session.cookie-domain
session.cookie_domain=
; Whether or not to add the httpOnly flag to the cookie, which makes it
; inaccessible to browser scripting languages such as JavaScript.
; http://php.net/session.cookie-httponly
session.cookie_httponly=
; Add SameSite attribute to cookie to help mitigate Cross-Site Request Forgery (CSRF/XSRF)
; Current valid values are "Lax" or "Strict"
; https://tools.ietf.org/html/draft-west-first-party-cookies-07
session.cookie_samesite=
; Handler used to serialize data. php is the standard serializer of PHP.
; http://php.net/session.serialize-handler
session.serialize_handler=php
; Defines the probability that the 'garbage collection' process is started
; on every session initialization. The probability is calculated by using
; gc_probability/gc_divisor. Where session.gc_probability is the numerator
; and gc_divisor is the denominator in the equation. Setting this value to 1
; when the session.gc_divisor value is 100 will give you approximately a 1% chance
; the gc will run on any given request.
```

```
; Default Value: 1
; Development Value: 1
; Production Value: 1
; http://php.net/session.gc-probability
session.gc_probability=1
; Defines the probability that the 'garbage collection' process is started on every
; session initialization. The probability is calculated by using the following equation:
; gc_probability/gc_divisor. Where session.gc_probability is the numerator and
; session.gc_divisor is the denominator in the equation. Setting this value to 100
; when the session.gc_probability value is 1 will give you approximately a 1% chance
; the gc will run on any given request. Increasing this value to 1000 will give you
; a 0.1% chance the gc will run on any given request. For high volume production servers,
; this is a more efficient approach.
; Default Value: 100
; Development Value: 1000
; Production Value: 1000
; http://php.net/session.gc-divisor
session.gc_divisor=1000
; After this number of seconds, stored data will be seen as 'garbage' and
; cleaned up by the garbage collection process.
; http://php.net/session.gc-maxlifetime
session.gc_maxlifetime=1440
; NOTE: If you are using the subdirectory option for storing session files
;       (see session.save_path above), then garbage collection does *not*
;       happen automatically.  You will need to do your own garbage
;       collection through a shell script, cron entry, or some other method.
;       For example, the following script would is the equivalent of
;       setting session.gc_maxlifetime to 1440 (1440 seconds = 24 minutes):
;          find /path/to/sessions -cmin +24 -type f | xargs rm
; Check HTTP Referer to invalidate externally stored URLs containing ids.
; HTTP_REFERER has to contain this substring for the session to be
; considered as valid.
; http://php.net/session.referer-check
session.referer_check=
; Set to {nocache,private,public,} to determine HTTP caching aspects
; or leave this empty to avoid sending anti-caching headers.
; http://php.net/session.cache-limiter
```

```
session.cache_limiter=nocache

; Document expires after n minutes.

; http://php.net/session.cache-expire

session.cache_expire=180

; trans sid support is disabled by default.

; Use of trans sid may risk your users' security.

; Use this option with caution.

; - User may send URL contains active session ID

;   to other person via. email/irc/etc.

; - URL that contains active session ID may be stored

;   in publicly accessible computer.

; - User may access your site with the same session ID

;   always using URL stored in browser's history or bookmarks.

; http://php.net/session.use-trans-sid

session.use_trans_sid=0

; Set session ID character length. This value could be between 22 to 256.

; Shorter length than default is supported only for compatibility reason.

; Users should use 32 or more chars.

; http://php.net/session.sid-length

; Default Value: 32

; Development Value: 26

; Production Value: 26

session.sid_length=26

; The URL rewriter will look for URLs in a defined set of HTML tags.

; <form> is special; if you include them here, the rewriter will

; add a hidden <input> field with the info which is otherwise appended

; to URLs. <form> tag's action attribute URL will not be modified

; unless it is specified.

; Note that all valid entries require a "=", even if no value follows.

; Default Value: "a=href,area=href,frame=src,form="

; Development Value: "a=href,area=href,frame=src,form="

; Production Value: "a=href,area=href,frame=src,form="

; http://php.net/url-rewriter.tags

session.trans_sid_tags="a=href,area=href,frame=src,form="

; URL rewriter does not rewrite absolute URLs by default.

; To enable rewrites for absolute paths, target hosts must be specified

; at RUNTIME. i.e. use ini_set()
```

```
; <form> tags is special. PHP will check action attribute's URL regardless
; of session.trans_sid_tags setting.
; If no host is defined, HTTP_HOST will be used for allowed host.
; Example value: php.net,www.php.net,wiki.php.net
; Use "," for multiple hosts. No spaces are allowed.
; Default Value: ""
; Development Value: ""
; Production Value: ""
;session.trans_sid_hosts=""
; Define how many bits are stored in each character when converting
; the binary hash data to something readable.
; Possible values:
;   4  (4 bits: 0-9, a-f)
;   5  (5 bits: 0-9, a-v)
;   6  (6 bits: 0-9, a-z, A-Z, "-", ",")
; Default Value: 4
; Development Value: 5
; Production Value: 5
; http://php.net/session.hash-bits-per-character
session.sid_bits_per_character=5
; Enable upload progress tracking in $_SESSION
; Default Value: On
; Development Value: On
; Production Value: On
; http://php.net/session.upload-progress.enabled
;session.upload_progress.enabled = On
; Cleanup the progress information as soon as all POST data has been read
; (i.e. upload completed).
; Default Value: On
; Development Value: On
; Production Value: On
; http://php.net/session.upload-progress.cleanup
;session.upload_progress.cleanup = On
; A prefix used for the upload progress key in $_SESSION
; Default Value: "upload_progress_"
; Development Value: "upload_progress_"
; Production Value: "upload_progress_"
```

```
; http://php.net/session.upload-progress.prefix

;session.upload_progress.prefix = "upload_progress_"

; The index name (concatenated with the prefix) in $_SESSION

; containing the upload progress information

; Default Value: "PHP_SESSION_UPLOAD_PROGRESS"

; Development Value: "PHP_SESSION_UPLOAD_PROGRESS"

; Production Value: "PHP_SESSION_UPLOAD_PROGRESS"

; http://php.net/session.upload-progress.name

;session.upload_progress.name = "PHP_SESSION_UPLOAD_PROGRESS"

; How frequently the upload progress should be updated.

; Given either in percentages (per-file), or in bytes

; Default Value: "1%"

; Development Value: "1%"

; Production Value: "1%"

; http://php.net/session.upload-progress.freq

;session.upload_progress.freq =  "1%"

; The minimum delay between updates, in seconds

; Default Value: 1

; Development Value: 1

; Production Value: 1

; http://php.net/session.upload-progress.min-freq

;session.upload_progress.min_freq = "1"

; Only write session data when session data is changed. Enabled by default.

; http://php.net/session.lazy-write

;session.lazy_write = On

[Assertion]

; Switch whether to compile assertions at all (to have no overhead at run-time)

; -1: Do not compile at all

;  0: Jump over assertion at run-time

;  1: Execute assertions

; Changing from or to a negative value is only possible in php.ini! (For turning assertions on and off at run-time, see assert.active, when
zend.assertions = 1)

; Default Value: 1

; Development Value: 1

; Production Value: -1

; http://php.net/zend.assertions

zend.assertions=1
```

```
; Assert(expr); active by default.

; http://php.net/assert.active

;assert.active = On

; Throw an AssertionError on failed assertions

; http://php.net/assert.exception

;assert.exception = On

; Issue a PHP warning for each failed assertion. (Overridden by assert.exception if active)

; http://php.net/assert.warning

;assert.warning = On

; Don't bail out by default.

; http://php.net/assert.bail

;assert.bail = Off

; User-function to be called if an assertion fails.

; http://php.net/assert.callback

;assert.callback = 0

; Eval the expression with current error_reporting().  Set to true if you want

; error_reporting(0) around the eval().

; http://php.net/assert.quiet-eval

;assert.quiet_eval = 0

[COM]

; path to a file containing GUIDs, IIDs or filenames of files with TypeLibs

; http://php.net/com.typelib-file

;com.typelib_file =

; allow Distributed-COM calls

; http://php.net/com.allow-dcom

;com.allow_dcom = true

; autoregister constants of a component's typlib on com_load()

; http://php.net/com.autoregister-typelib

;com.autoregister_typelib = true

; register constants casesensitive

; http://php.net/com.autoregister-casesensitive

;com.autoregister_casesensitive = false

; show warnings on duplicate constant registrations

; http://php.net/com.autoregister-verbose

;com.autoregister_verbose = true

; The default character set code-page to use when passing strings to and from COM objects.

; Default: system ANSI code page
```

```
;com.code_page=

[mbstring]

; language for internal character representation.

; This affects mb_send_mail() and mbstring.detect_order.

; http://php.net/mbstring.language

;mbstring.language = Japanese

; Use of this INI entry is deprecated, use global internal_encoding instead.

; internal/script encoding.

; Some encoding cannot work as internal encoding. (e.g. SJIS, BIG5, ISO-2022-*)

; If empty, default_charset or internal_encoding or iconv.internal_encoding is used.

; The precedence is: default_charset < internal_encoding < iconv.internal_encoding

;mbstring.internal_encoding =

; Use of this INI entry is deprecated, use global input_encoding instead.

; http input encoding.

; mbstring.encoding_translation = On is needed to use this setting.

; If empty, default_charset or input_encoding or mbstring.input is used.

; The precedence is: default_charset < input_encoding < mbsting.http_input

; http://php.net/mbstring.http-input

;mbstring.http_input =

; Use of this INI entry is deprecated, use global output_encoding instead.

; http output encoding.

; mb_output_handler must be registered as output buffer to function.

; If empty, default_charset or output_encoding or mbstring.http_output is used.

; The precedence is: default_charset < output_encoding < mbstring.http_output

; To use an output encoding conversion, mbstring's output handler must be set

; otherwise output encoding conversion cannot be performed.

; http://php.net/mbstring.http-output

;mbstring.http_output =

; enable automatic encoding translation according to

; mbstring.internal_encoding setting. Input chars are

; converted to internal encoding by setting this to On.

; Note: Do _not_ use automatic encoding translation for

;       portable libs/applications.

; http://php.net/mbstring.encoding-translation

;mbstring.encoding_translation = Off

; automatic encoding detection order.

; "auto" detect order is changed according to mbstring.language
```

```
; http://php.net/mbstring.detect-order

;mbstring.detect_order = auto

; substitute_character used when character cannot be converted

; one from another

; http://php.net/mbstring.substitute-character

;mbstring.substitute_character = none

; overload(replace) single byte functions by mbstring functions.

; mail(), ereg(), etc are overloaded by mb_send_mail(), mb_ereg(),

; etc. Possible values are 0,1,2,4 or combination of them.

; For example, 7 for overload everything.

; 0: No overload

; 1: Overload mail() function

; 2: Overload str*() functions

; 4: Overload ereg*() functions

; http://php.net/mbstring.func-overload

;mbstring.func_overload = 0

; enable strict encoding detection.

; Default: Off

;mbstring.strict_detection = On

; This directive specifies the regex pattern of content types for which mb_output_handler()

; is activated.

; Default: mbstring.http_output_conv_mimetype=^(text/|application/xhtml\+xml)

;mbstring.http_output_conv_mimetype=

; This directive specifies maximum stack depth for mbstring regular expressions. It is similar

; to the pcre.recursion_limit for PCRE.

; Default: 100000

;mbstring.regex_stack_limit=100000

[gd]

; Tell the jpeg decode to ignore warnings and try to create

; a gd image. The warning will then be displayed as notices

; disabled by default

; http://php.net/gd.jpeg-ignore-warning

;gd.jpeg_ignore_warning = 1

[exif]

; Exif UNICODE user comments are handled as UCS-2BE/UCS-2LE and JIS as JIS.

; With mbstring support this will automatically be converted into the encoding

; given by corresponding encode setting. When empty mbstring.internal_encoding
```

```
; is used. For the decode settings you can distinguish between motorola and
; intel byte order. A decode setting cannot be empty.
; http://php.net/exif.encode-unicode
;exif.encode_unicode = ISO-8859-15
; http://php.net/exif.decode-unicode-motorola
;exif.decode_unicode_motorola = UCS-2BE
; http://php.net/exif.decode-unicode-intel
;exif.decode_unicode_intel    = UCS-2LE
; http://php.net/exif.encode-jis
;exif.encode_jis =
; http://php.net/exif.decode-jis-motorola
;exif.decode_jis_motorola = JIS
; http://php.net/exif.decode-jis-intel
;exif.decode_jis_intel    = JIS
[Tidy]
; The path to a default tidy configuration file to use when using tidy
; http://php.net/tidy.default-config
;tidy.default_config = /usr/local/lib/php/default.tcfg
; Should tidy clean and repair output automatically?
; WARNING: Do not use this option if you are generating non-html content
; such as dynamic images
; http://php.net/tidy.clean-output
tidy.clean_output=Off
[soap]
; Enables or disables WSDL caching feature.
; http://php.net/soap.wsdl-cache-enabled
soap.wsdl_cache_enabled=1
; Sets the directory name where SOAP extension will put cache files.
; http://php.net/soap.wsdl-cache-dir
soap.wsdl_cache_dir="/tmp"
; (time to live) Sets the number of second while cached file will be used
; instead of original one.
; http://php.net/soap.wsdl-cache-ttl
soap.wsdl_cache_ttl=86400
; Sets the size of the cache limit. (Max. number of WSDL files to cache)
soap.wsdl_cache_limit=5
[sysvshm]
```

```
; A default size of the shared memory segment

;sysvshm.init_mem = 10000

[ldap]

; Sets the maximum number of open links or -1 for unlimited.

ldap.max_links=-1

[dba]

;dba.default_handler=

[opcache]

; Determines if Zend OPCache is enabled

;opcache.enable=1

; Determines if Zend OPCache is enabled for the CLI version of PHP

;opcache.enable_cli=0

; The OPcache shared memory storage size.

;opcache.memory_consumption=128

; The amount of memory for interned strings in Mbytes.

;opcache.interned_strings_buffer=8

; The maximum number of keys (scripts) in the OPcache hash table.

; Only numbers between 200 and 1000000 are allowed.

;opcache.max_accelerated_files=10000

; The maximum percentage of "wasted" memory until a restart is scheduled.

;opcache.max_wasted_percentage=5

; When this directive is enabled, the OPcache appends the current working

; directory to the script key, thus eliminating possible collisions between

; files with the same name (basename). Disabling the directive improves

; performance, but may break existing applications.

;opcache.use_cwd=1

; When disabled, you must reset the OPcache manually or restart the

; webserver for changes to the filesystem to take effect.

;opcache.validate_timestamps=1

; How often (in seconds) to check file timestamps for changes to the shared

; memory storage allocation. ("1" means validate once per second, but only

; once per request. "0" means always validate)

;opcache.revalidate_freq=2

; Enables or disables file search in include_path optimization

;opcache.revalidate_path=0

; If disabled, all PHPDoc comments are dropped from the code to reduce the

; size of the optimized code.
```

```
;opcache.save_comments=1

; Allow file existence override (file_exists, etc.) performance feature.

;opcache.enable_file_override=0

; A bitmask, where each bit enables or disables the appropriate OPcache

; passes

;opcache.optimization_level=0x7FFFBFFF

;opcache.dups_fix=0

; The location of the OPcache blacklist file (wildcards allowed).

; Each OPcache blacklist file is a text file that holds the names of files

; that should not be accelerated. The file format is to add each filename

; to a new line. The filename may be a full path or just a file prefix

; (i.e., /var/www/x  blacklists all the files and directories in /var/www

; that start with 'x'). Line starting with a ; are ignored (comments).

;opcache.blacklist_filename=

; Allows exclusion of large files from being cached. By default all files

; are cached.

;opcache.max_file_size=0

; Check the cache checksum each N requests.

; The default value of "0" means that the checks are disabled.

;opcache.consistency_checks=0

; How long to wait (in seconds) for a scheduled restart to begin if the cache

; is not being accessed.

;opcache.force_restart_timeout=180

; OPcache error_log file name. Empty string assumes "stderr".

;opcache.error_log=

; All OPcache errors go to the Web server log.

; By default, only fatal errors (level 0) or errors (level 1) are logged.

; You can also enable warnings (level 2), info messages (level 3) or

; debug messages (level 4).

;opcache.log_verbosity_level=1

; Preferred Shared Memory back-end. Leave empty and let the system decide.

;opcache.preferred_memory_model=

; Protect the shared memory from unexpected writing during script execution.

; Useful for internal debugging only.

;opcache.protect_memory=0

; Allows calling OPcache API functions only from PHP scripts which path is

; started from specified string. The default "" means no restriction
```

```
;opcache.restrict_api=

; Mapping base of shared memory segments (for Windows only). All the PHP

; processes have to map shared memory into the same address space. This

; directive allows to manually fix the "Unable to reattach to base address"

; errors.

;opcache.mmap_base=

; Enables and sets the second level cache directory.

; It should improve performance when SHM memory is full, at server restart or

; SHM reset. The default "" disables file based caching.

;opcache.file_cache=

; Enables or disables opcode caching in shared memory.

;opcache.file_cache_only=0

; Enables or disables checksum validation when script loaded from file cache.

;opcache.file_cache_consistency_checks=1

; Implies opcache.file_cache_only=1 for a certain process that failed to

; reattach to the shared memory (for Windows only). Explicitly enabled file

; cache is required.

;opcache.file_cache_fallback=1

; Enables or disables copying of PHP code (text segment) into HUGE PAGES.

; This should improve performance, but requires appropriate OS configuration.

;opcache.huge_code_pages=0

; Validate cached file permissions.

;opcache.validate_permission=0

; Prevent name collisions in chroot'ed environment.

;opcache.validate_root=0

; If specified, it produces opcode dumps for debugging different stages of

; optimizations.

;opcache.opt_debug_level=0

[curl]

; A default value for the CURLOPT_CAINFO option. This is required to be an

; absolute path.

curl.cainfo="C:\xampp\apache\bin\curl-ca-bundle.crt"

[openssl]

; The location of a Certificate Authority (CA) file on the local filesystem

; to use when verifying the identity of SSL/TLS peers. Most users should

; not specify a value for this directive as PHP will attempt to use the

; OS-managed cert stores in its absence. If specified, this value may still
```

```
; be overridden on a per-stream basis via the "cafile" SSL stream context
; option.
openssl.cafile="C:\xampp\apache\bin\curl-ca-bundle.crt"
; If openssl.cafile is not specified or if the CA file is not found, the
; directory pointed to by openssl.capath is searched for a suitable
; certificate. This value must be a correctly hashed certificate directory.
; Most users should not specify a value for this directive as PHP will
; attempt to use the OS-managed cert stores in its absence. If specified,
; this value may still be overridden on a per-stream basis via the "capath"
; SSL stream context option.
;openssl.capath=
; Local Variables:
; tab-width: 4
; End:
[Syslog]
define_syslog_variables=Off
[Session]
define_syslog_variables=Off
[Date]
date.timezone=Europe/Berlin
[MySQL]
mysql.allow_local_infile=On
mysql.allow_persistent=On
mysql.cache_size=2000
mysql.max_persistent=-1
mysql.max_link=-1
mysql.default_port=3306
mysql.default_socket="MySQL"
mysql.connect_timeout=3
mysql.trace_mode=Off
[Sybase-CT]
sybct.allow_persistent=On
sybct.max_persistent=-1
sybct.max_links=-1
sybct.min_server_severity=10
sybct.min_client_severity=10
[MSSQL]
```

```
mssql.allow_persistent=On

mssql.max_persistent=-1

mssql.max_links=-1

mssql.min_error_severity=10

mssql.min_message_severity=10

mssql.compatability_mode=Off

mssql.secure_connection=Off
```

view rawphp.ini hosted with ❤ by GitHub

Nota: quando si effettuano modifiche al file php.ini bisogna ricordarsi di riavviare il web server per rendere effettive le modifiche. Se si sta utilizzando XAMPP aprire il pannello di controllo, stoppare Apache e riavviarlo.

Di seguito sono elencate alcune regole di sintassi per le direttive:

- Ciascuna direttiva è composta dal nome seguito dal simbolo di uguale e dal valore.
- Le direttive sono *case sensitive,* i valori no.
- Le righe che iniziano con il punto e virgola vengono ignorate.
- I valori possono essere stringhe, numeri, costanti, oppure valori booleani, che indicano l'attivazione/disattivazione di una direttiva: 1/0, yes/no, on/off o true/false.

Vediamo alcune delle direttive più usate e il loro significato:

Direttiva PHP	Descrizione
allow_url_include	Consente l'utilizzo di wrapper fopen compatibili con l'URL con le seguenti funzioni: include(), include_once(), require(), require_once().
disable_functions	Consente di disabilitare determinate funzioni per motivi di sicurezza. Ciascun nome di funzione è delimitato da una virgola. Es. show_source, system, shell_exec, passthru, exec, popen, proc_open
display_errors	Determina se gli errori devono essere mostrati a video o se devono essere nascosti all'utente.
file_uploads	Determina se consentire o meno il caricamento di file.
max_execution_time	Imposta il tempo massimo in secondi di esecuzione di uno script, prima che venga terminato dal parser. Ciò impedisce che programmi scritti male possano bloccare il web server. Il valore ottimale è 30.
memory_limit	Imposta la quantità massima di memoria RAM in byte che uno script è autorizzato ad allocare. Questo previene che la memoria venga saturata da un programma scritto male. Un valore pari a -1 non pone limiti all'utilizzo di memoria. È possibile utilizzare il suffisso M davanti al valore per indicare i megabyte. Es. 128M.
post_max_size	Imposta la dimensione massima consentita per i dati inviati tramite POST. Questa impostazione influenza anche il caricamento dei file, dato che questi vengono inviati al server tramite una richiesta POST. Per caricare file di grandi dimensioni, questo valore deve essere maggiore di upload_max_filesize.
session.gc_maxlifetime	Specifica la scadenza in secondi dei dati di sessione. Oltrepassato questo valore i dati verranno ripuliti.
upload_max_filesize	Specifica la dimensione massima che può avere un singolo file prima di essere caricato.

La lista completa delle direttive PHP può essere consultata sul sito ufficiale all'indirizzo **https://www.php.net/manual/en/ini.php**.

Estensioni del linguaggio

Esistono alcune direttive che consentono di attivare/disattivare le **estensioni del linguaggio**, chiamate anche *moduli*, che permettono di arricchire le funzionalità di PHP.

Alcune delle estensioni più usate:

- extension=curl: abilita l'utilizzo di cURL per il trasferimento di dati su vari protocolli.
- extension=fileinfo: abilita la libreria FileInfo, che consente di ricavare tipo di contenuto e codifica di un file, analizzando determinate sequenze di byte all'interno del file.
- extension=gd2: abilita la libreria GD2 per la manipolazione di immagini.
- extension=mbstring: abilita il supporto per le stringhe multi-byte.
- extension=mysqli: fornisce un'interfaccia per poter interagire con MySQL. È una versione migliorata dell'ormai rimossa estensione mysql.
- extension=openssl: abilita l'uso della libreria OpenSSL per la crittografia simmetrica/asimmetrica.
- extension=pdo_mysql: abilita l'utilizzo della libreria PHP Data Objects (PDO), per l'accesso al database MySQL tramite un'unica interfaccia.

Tramite le Zend API, ossia l'interfaccia di comunicazione verso l'**interprete PHP**, è possibile scrivere ulteriori estensioni che estendono le funzionalità del linguaggio. È necessaria la conoscenza del linguaggio di programmazione C.

È possibile visualizzare la lista completa delle estensioni attive tramite la funzione PHP **get_loaded_extensions()**.

Verificare la configurazione PHP

È possibile verificare i cambiamenti alla configurazione PHP digitando il comando php -info dal terminale di sistema.

Un metodo alternativo consiste nell'utilizzo della funzione nativa phpinfo(), posta all'interno di un file .php.

```
<?php




phpinfo();
```

Per ottenere l'output della configurazione PHP segui i passi seguenti:

1. Apri la directory C:\xampp\htdocs
2. Crea un file di testo e rinominalo in phpinfo.php
3. Apri il file con un editor di testo e scrivi il codice dell'esempio, quindi salva.
4. Apri il browser e visualizza l'output del programma all'indirizzo http://localhost/phpinfo.php

L'output del programma sarà diverso a seconda del sistema operativo in uso e dei settaggi nel php.ini.

PHP Version 7.3.5

System	Windows NT ZUCCHINO-PC 6.1 build 7601 (Windows 7 Professional Edition Ser
Build Date	May 1 2019 13:10:59
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--wi build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,TS,VC15
PHP Extension Build	API20180731,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled

Ora che abbiamo installato e configurato PHP, possiamo passare alla guida successiva dove introdurremo la **sintassi PHP** e vedremo i primi esempi di codice.

Sintassi PHP

Prima di dare uno sguardo alla **sintassi PHP**, spieghiamo come poter scrivere e visualizzare l'output di un programma scritto in questo linguaggio:

- Se hai installato XAMPP, i tuoi script vanno salvati nella cartella htdocs. Il percorso della cartella è C:\xampp\htdocs.
- Apri il tuo editor preferito, inserisci il codice PHP (**vedi esempio in basso**).
- Quindi salva il tuo file con estensione .php all'interno della cartella htdocs. Es. test.php
- Apri il tuo browser preferito e visita l'indirizzo http://localhost/test.php.
- Quindi se ad esempio il tuo file si trova: C:\xampp\htdocs\test.php, devi digitare nel browser http: //localhost/test.php.
- Il tuo programma PHP verrà eseguito e fornirà l'output in HTML.

Per maggiori dettagli visita la **guida come installare XAMPP**.

Un'altra modalità di esecuzione consiste nel far eseguire direttamente il programma dall'interprete PHP.

È possibile fare ciò lanciando da terminale (*cmd* per Windows o la *shell* per Linux) il seguente comando

```
php test.php
```

Adesso vediamo come deve essere formattato il codice PHP all'interno del nostro file e come utilizzarlo in combinazione con HTML. Per ultimo daremo uno sguardo ai commenti PHP.

Indice dei contenuti

1. **I tag PHP**
2. **PHP e HTML**
3. **Commenti PHP**

I tag PHP

Qualsiasi script PHP inizia con il tag di apertura <?php e termina con il tag di chiusura ?>(sono detti **tag PHP canonici**), come mostra il seguente esempio:

```
<?php

echo "Hello World!";

?>
```

L'esempio appena visto stampa a video la classica scritta Hello World! tramite il costrutto echo del linguaggio. I doppi apici "" indicano una stringa, ovvero una sequenza di caratteri.

> **Nota:** Se il file contiene solo codice PHP, è consigliabile omettere il tag di chiusura. Questo impedisce che spazi vuoti accidentali o nuove righe vengano aggiunte dopo il tag di chiusura, il che può causare errori in fase di esecuzione poiché PHP avvierà il buffering dell'output senza intenzione da parte del programmatore.

Ciascuna riga di codice termina con un punto e virgola, che indica all'**interprete PHP** la fine dell'istruzione. Si può omettere il punto e virgola solo nel caso dell'ultima istruzione di un blocco di codice racchiuso tra entrambi i tag PHP, altrimenti viene generato un **errore di sintassi** come nel seguente esempio:

```
<?php

echo "Hello World!" // Corretto!

?>



<?php

echo "Hello World!" // Errore! È necessario il ; oppure il tag di chiusura
```

> **Nota:** PHP consente anche l'utilizzo del tag di apertura breve <?, tuttavia se ne sconsiglia l'uso in quanto è disponibile solo se è abilitata la direttiva short_open_tag del file di configurazione php.ini. Inoltre da **PHP 7.4**, il tag di apertura breve verrà deprecato e dunque rimosso nella prossima versione.

PHP e HTML

All'interno di un file .php è possibile incorporare **codice PHP e HTML**. Per questo motivo si dice che PHP é un linguaggio HTML-embedded.

```
<html>

    <body>

        <strong>

            <?php

            echo "Hello World!";

            ?>

        </strong>
```

```
    </body>

</html>
```

Il codice sopra riportato stampa in grassetto la scritta Hello World!. Se visualizziamo il codice sorgente attraverso il browser, noteremo che il codice PHP é stato sostituito da codice HTML.

```
<html>

    <body>

        <strong>

            Hello World!

        </strong>

    </body>

</html>
```

In che modo avviene tutto questo?

Quando l'interprete PHP analizza il codice, cerca i **tag di apertura e di chiusura** ed esegue solamente le istruzioni incluse tra tali tag, ignorando il resto del codice presente nel file. Alla fine il web server invia l'output al browser che visualizza l'HTML risultante.

Nota Anche se è possibile mescolare insieme PHP e HTML, è buona prassi separare i due linguaggi in file diversi. HTML infatti è un linguaggio di markup che serve a formattare il testo, PHP invece effettua elaborazioni sui dati. Dunque PHP dovrebbe essere incluso nell'HTML solamente per stampare il contenuto di variabili.

Commenti PHP

Un **commento PHP** è un semplice testo che viene ignorato dall'interprete del linguaggio. Il suo scopo è quello di rendere il codice più leggibile e aiutare l'autore del codice, o altri sviluppatori, a capire meglio cosa fa il programma anche a distanza di tempo.

Commentare codice PHP è un operazione semplice. Il linguaggio mette a disposizione due formati di commenti:

1. **Commenti a riga singola**. Generalmente vengono utilizzati per brevi spiegazioni o note rilevanti per il codice. Per inserirli è sufficiente far precedere il testo del commento da un doppio slash // oppure dal simbolo del cancelletto #.
2. **Commenti multi riga**. Servono a fornire spiegazioni più dettagliate sul codice PHP. In questo caso il testo del commento viene racchiuso tra /* e */.

```
<?php


// Questo è un commento


# Questo è un altro commento






/*


* Autore: Flavio


* Linguaggio: PHP


* Oggetto: Commento su più righe


*/






echo "Esempi di commenti PHP";
?>
```

Dopo aver visto come scrivere un semplice programma PHP, passiamo alla guida successiva che descrive una delle caratteristiche più importanti di un linguaggio di programmazione: le variabili e i tipi di dati.

Variabili e tipi di dati in PHP

Le variabili sono una delle strutture fondamentali della programmazione. Vengono utilizzate per **memorizzare dati**, come stringhe di testo e numeri.

Come quasi tutti i linguaggi di programmazione anche PHP utilizza le variabili per consentire l'elaborazione dei dati.

Una variabile in PHP è una **locazione di memoria temporanea** dove sono memorizzate delle informazioni, di conseguenza il ciclo di vita di una variabile dura fino al termine dell'esecuzione del programma.

Creazione

Una **variabile in PHP** è costituita da due parti: il **nome** e il **valore**.

Come altri linguaggi anche PHP prevede delle regole per la denominazione delle variabili:

- Tutte le variabili iniziano con il **simbolo del dollaro** $, seguito dal nome della variabile.
- Il nome della variabile deve iniziare con una lettera o un underscore _, i restanti caratteri possono essere lettere, numeri o underscore.
- Il nome della variabile non può cominciare con un numero e non può contenere spazi.

L'esempio seguente mostra nomi di variabili in PHP perfettamente validi:

```
<?php

$name = 'Flavio';

$age = 36;

$_country = 'Italia';

$car1 = 'Clio';



echo "Mi chiamo $name, vivo in $_country, ho $age anni e guido una $car1";



// Mi chiamo Flavio, vivo in Italia, ho 36 anni e guido una Clio

?>
```

I nomi di variabili sono ***case sensitive***, quindi $name e $Name sono due variabili distinte e rappresentano due locazioni di memoria differenti.

È consigliabile assegnare alle variabili un nome comprensibile che ne descriva il comportamento all'interno del programma.

Se il nome della variabile è composto da due o più parole (Es. "is checked" oppure "my first name"), per convenzione si utilizza la **notazione camelCase**, quindi presi i due esempi in parentesi avremo $isChecked e $myFirstName.

Quando si crea una variabile è importante considerare anche il **tipo di dato** che le si assegna, ovvero il tipo di contenuto della variabile.

Visibilità

La visibilità (o scope) di una variabile indica il contesto in cui essa è accessibile all'interno di un programma.

In PHP la maggior parte delle variabili ha uno **scope globale**, ovvero è raggiungibile in qualsiasi punto del programma. Le variabili contenute all'interno di funzioni hanno invece visibilità locale, nel senso che possono essere utilizzate solo all'interno di quella funzione.

Vedremo nel dettaglio l'argomento scope variabili quando parleremo di **funzioni PHP**.

Casting

In un linguaggio di programmazione, l'esecuzione di calcoli aritmetici richiede che le variabili siano dello stesso tipo di dati. Con PHP questo non è necessario, in quanto **l'interprete del linguaggio** si occupa di effettuare il casting in automatico.

Fare il **casting di una variabile** vuol dire convertire una variabile o un valore da un tipo di dato a un altro.

Nonostante questo processo in PHP sia automatizzato, esiste la possibilità di fare un casting esplicito. Vediamo meglio con un esempio

```
<?php
$var1 = 10;
$var2 = 6.5;
var_dump($var1 + $var2); // float(16.5)
$var2 = (int) 6.5;
var_dump($var1 + $var2); // int(16)
?>
```

Nell'esempio precedente abbiamo eseguito, tramite l'operatore aritmetico +, la somma delle variabili $var1 e $var2, il cui tipo è rispettivamente **integer** (numero intero) e **float** (virgola mobile).

Il risultato della somma viene mostrato a video tramite la funzione var_dump(), che stampa sia il valore che il tipo di dato.

Alla riga 7 facciamo un **casting esplicito** a integer della variabile $var2 indicando tra parentesi tonde la keyword int. Questo converte il tipo da float e integer. Il risultato mostrato alla riga 9 è un valore di tipo integer.

Nelle prossime guide, vedremo in dettaglio i **tipi di dati** che PHP è in grado di gestire e li analizzeremo singolarmente.

Tipo boolean in PHP

Capita spesso nei programmi PHP, di dover memorizzare in una **variabile** se un determinato evento è accaduto o meno. Ad esempio se un utente ha effettuato l'accesso ad un'area riservata.

In questo caso, PHP mette a disposizione il tipo di dato **boolean** (o booleano). Una variabile di tipo booleano può avere solo due valori: true o false.

```
<?php

$isLogged = true;



if ($isLogged) {

    echo "Loggato";

} else {

    echo "Non loggato";

}

// Output: Loggato

?>
```

Nell'esempio precedente abbiamo creato una variabile $isLogged di tipo booleano semplicemente assegnandole il valore true. Nelle righe da 4 a 8 abbiamo utilizzato l'istruzione if-elseche spiegheremo nel dettaglio quando parleremo di **istruzioni condizionali**.

In breve, nell'istruzione if viene valutata l'espressione condizionale posta tra parentesi tonde: se questa è uguale a true, viene eseguita la prima porzione di codice contenuta tra parentesi graffe, altrimenti viene eseguita la seconda porzione subito dopo l'istruzione else.

> **Nota:** true è una **costante PHP**, così come false. Per convenzione le costanti vengono indicate in maiuscolo, ma in questo caso si usa il minuscolo.

Tipi di dati interpretati come booleani

PHP ha un meccanismo interno di **conversione dei tipi** di dati diversi dal booleano.

Prendendo spunto dall'esempio precedente, assegnamo alla variabile `$isLogged` il valore di tipo integer `1` .

Nel momento in cui valuta l'espressione condizionale, l'interprete PHP converte il tipo di dato da **integer** a booleano (in questo caso`true`) e viene stampato `Loggato`. Lo stesso succede quando valorizziamo la variabile con la stringa `"Loggato"`.

```
<?php
$isLogged = 1;
if ($isLogged) {
    echo "Loggato\n";


} else {


    echo "Non loggato\n";


}


// Output: Loggato


$isLogged = "Loggato";


if ($isLogged) {


    echo "Loggato";


} else {


    echo "Non loggato";


}


// Output: Loggato


?>
```

In questo esempio abbiamo usato la sequenza di escape \n per andare a capo. Vedremo questa e altre sequenze di escape quando parleremo del tipo **string**.

La tabella seguente mostra come vengono interpretati i valori non booleani nelle espressioni condizionali.

Valore	Interpretazione
integer	È false solo se uguale a zero 0, true altrimenti
string	È false se la stringa è vuota "" oppure è uguale a "0", true altrimenti.
array	È false solo se non contiene valori (array vuoto), true altrimenti.
null	È sempre false.
float	È false solo se uguale a 0.0, true altrimenti.
object	È false se l'oggetto è null, true altrimenti.

Tipo integer in PHP

Il tipo integer in PHP è un **numero intero** positivo o negativo. PHP memorizza questi numeri con 32 o 64 bit a seconda del sistema operativo utilizzato.

Tramite le costanti PHP_INT_MAX e PHP_INT_MINpossiamo verificare rispettivamente il numero massimo e il numero minimo supportati da PHP.

È possibile rappresentare un numero intero in quattro diverse basi:

- Base 10 (intero decimale).
- Base 8 (intero ottale). Specificato tramite il prefisso 0 seguito dal valore del numero.
- Base 16 (intero esadecimale). Specificato tramite il prefisso 0x seguito dal valore del numero.
- Base 2 (intero binario). Specificato tramite il prefisso 0b seguito dal valore del numero.

Il seguente esempio mostra le diverse rappresentazioni dei numeri interi:

```
<?php

$dec = 40; // decimale

$oct = 0321; // ottale

$hex = 0xBC; // esadecimale

$bin = 0b11111; // binario

$neg = -125; // negativo



echo "$dec\n"; // 40

echo "$oct\n"; // 209

echo "$hex\n"; // 188

echo "$bin\n"; // 31

echo "$neg\n"; // -125
?>
```

Tipo float in PHP

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.

Vediamo subito un esempio

```
<?php

$float1 = 123.30; // Sintassi con il punto

$float2 = 120e-3; // Sintassi con esponente

echo "$float1\n"; // 123.30

echo "$float2\n"; // 0.12
?>
```

La sintassi con l'esponente consente di specificare numeri elevati alla potenza di 10. Nell'esempio 120e-3 equivale a $120 * 10^{-3}$.

Nota: Quando si effettua una divisione tra due numeri, il risultato viene rappresentato internamente da PHP come un float, anche se è un numero intero.

Funzioni per il calcolo in virgola mobile

PHP mette a disposizione alcune funzioni native per il calcolo in virgola mobile. Vediamone alcune:

Funzione	Descrizione	Esempio

ceil()	Restituisce il numero arrotondato all'intero superiore.	ceil(3.3); // 4
		
		ceil(4.99); // 5
		
		ceil(-8.16); // -8

floor()	Restituisce il numero arrotondato all'intero inferiore.	floor(20.3); // 20
		
		floor(6.50); // 6
		
		floor(-15.60); // -16

pi()	Restituisce il valore di PI greco.	pi(); // 3.1415926535898
sqrt()	Restituisce la radice quadrata di un numero.	sqrt(15) // 3.8729833462074

L'elenco completo delle funzioni matematiche è consultabile all'indirizzo **https://www.php.net/manual/en/ref.math.php**.

PHP mette a disposizione gli **operatori aritmetici** per effettuare le classiche operazioni matematiche sui numeri (addizione, sottrazione, moltiplicazione e divisione). Li analizzeremo in una delle prossime guide.

Tipo string in PHP

Il tipo string in PHP viene utilizzato per memorizzare **stringhe di caratteri** ASCII, dove ciascun carattere è rappresentato da un **byte**.

Una stringa può contenere lettere, numeri e caratteri speciali.

Il modo più semplice per creare una stringa è racchiuderla tra **singoli apici** ''. È anche possibile utilizzare i **doppi apici** "". Ad ogni modo singoli e doppi apici lavorano in modi differenti.

```
<?php

$car = "Ferrari";

$model = 'F40';



echo "Auto $car \n"; // Auto Ferrari

echo "Modello $model\n"; // Modello F40
```

```
echo 'Auto $car \n'; // Auto $car \n


echo 'Modello $model \n'; // Modello $model \n
```

L'esempio mostra come le stringhe tra singoli apici vengono interpretate così come sono, mentre quelle tra doppi apici vengono pre-elaborate da PHP nel seguente modo:

- I nomi delle variabili contenute nella stringa vengono sostituiti con il loro valore ($car e $model nell'esempio).
- Alcune sequenze di caratteri (**sequenze di escape**) che iniziano con un backslash \, vengono sostituite con caratteri speciali (Vedi tabella).

Sequenze di escape

Sequenza di escape	Descrizione
\n	Sostituito dal carattere di nuova riga.
\r	Sostituito dal carattere di carriage-return.
\t	Sostituito dal carattere di tabulazione.
\$	Sostituito dal simbolo stesso del dollaro.
\"	Sostituito da un singolo doppio apice.
\\	Sostituito da un singolo backslash

Stringhe come array di caratteri

Come anticipato in una **guida precedente**, il tipo di dato string in PHP è un tipo speciale di array. Come vedremo gli array sono una sequenza ordinata di valori, ciascuno dei quali accessibile tramite un indice.

Le stringhe possono essere viste come sequenze ordinate di caratteri, ciascuno dei quali associato ad un indice numerico che consente di accedere a quel carattere.

Un esempio chiarisce meglio come PHP memorizza le stringhe

```
<?php

$lastName = "Biscaldi";



echo $lastName[0]; // B

echo $lastName[3]; // c

echo $lastName[4]; // a
?>
```

Nota: il primo elemento di ogni array ha indice numerico pari a 0, e non 1 come ci si potrebbe aspettare.

Concatenare stringhe

PHP mette a disposizione un **operatore di stringa** (operatore di concatenazione), rappresentato dal carattere del punto ., che consente di concatenare più stringhe tra di loro.

Di seguito è mostrato un esempio:

```
<?php

$string1 = "Mario Rossi ";

$string2 = "vive a Roma ";

$string3 = "e fa il programmatore";



echo $string1 . $string2 . $string3; // Mario Rossi vive a Roma e fa il programmatore

?>
```

Manipolare le stringhe in PHP

PHP offre diverse funzioni per manipolare le stringhe, in questo paragrafo ne vediamo solamente quattro tra quelle più comuni.

Per una panoramica più ampia consulta l'apposita guida sulle **funzioni per gestire le stringhe**.

- strlen() restituisce la lunghezza di una stringa.
- strpos() restituisce la posizione di una parola all'interno di una stringa.
- substr() restituisce una sottostringa da una stringa.
- str_replace() sostituisce tutte le occorrenza di una parola con un'altra all'interno di una stringa.

```php
<?php

$string = "Ciao mondo!";

echo strlen($string)."\n"; // 11

echo strpos($string, 'mondo')."\n"; // 5

echo substr($string, 5, 5)."\n"; // mondo

echo str_replace('mondo', 'a tutti', $string)."\n"; // Ciao a tutti
?>
```

Nell'output della funzione strpos() possiamo notare che la posizione di "mondo" nella nostra stringa è 5. Il motivo per cui è 5, e non 6, è perché negli array la prima posizione parte da 0 e non da 1, come ci si potrebbe aspettare. Questo vale per tutti i linguaggi di programmazione.

Costanti in PHP

Le **costanti in PHP** sono contenitori di valori come le **variabili**. A differenza di queste però, **possono essere definite una volta sola** all'interno del programma e il loro valore non può essere più modificato.

Per la definizione di costanti valgono le seguenti regole:

- Sono *case sensitive.*
- Possono contenere solo valori scalari: booleani, interi, numeri a virgola mobile e stringhe.
- Il nome della costante deve iniziare con una lettera o un underscore, i restanti caratteri possono essere lettere, numeri o underscore.
- Vengono definite tramite la funzione define(), che accetta due argomenti: il nome della costante e il suo valore.

Vediamo un esempio di utilizzo

```
<?php

define('SOFTWARE_VERSION', 7.2);

echo "Versione del software ".SOFTWARE_VERSION; // Versione del software 7.2
?>
```

Qui abbiamo introdotto l'operatore di concatenamento, rappresentato dal punto, utile quando dobbiamo unire due o più stringhe tra loro.

> **Nota:** Per convenzione i nomi delle costanti sono scritti in lettere maiuscole. Questo aiuta la loro identificazione, differenziandole dalle variabili all'interno del programma.

Differenze tra costanti e variabili

- Le variabili devono iniziare con il simbolo del dollaro $, le costanti non necessitato di questo.
- Le variabili vengono definite con una semplice assegnazione, le costanti necessitano della funzione define().
- L'ambito delle variabili è locale solo all'interno delle funzioni, altrimenti è globale. L'ambito delle costanti invece è globale in qualunque parte del programma.
- Una volta impostate, le costanti non possono essere ridefinite o modificate.

Costanti magiche

PHP mette a disposizione alcune costanti predefinite chiamate **costanti magiche**, la cui sintassi è la seguente: __NAME__.

Nome	Descrizione
__LINE__	Restituisce il numero della riga corrente di codice all'interno del file.
__FILE__	Restituisce il percorso completo e il nome del file che contiene lo script PHP.
__FUNCTION__	Restituisce il nome della funzione.
__CLASS__	Restituisce il nome della classe.
__METHOD__	Restituisce il nome del metodo della classe

Array PHP

Fino ad ora abbiamo visto **tipi di dati** semplici: booleani, interi, float e stringhe. PHP è in grado di gestire anche tipi di dati "composti", tra cui gli array.

Gli **array in PHP sono sequenze di valori di qualsiasi tipo** chiamati *elementi* e accessibili tramite un *indice*. Sono una delle strutture dati più usate nella programmazione.

Gli array possono essere utilizzati per costruire **strutture dati complesse** come liste, hash table, pile, code.

Indice dei contenuti

1. **Tipologie di array**
2. **Inizializzare un array**
3. **Accedere ad un elemento**
4. **Aggiungere un elemento**
5. **Modificare un elemento**
6. **Eliminare un elemento**
7. **Stampare un array**

Tipologie di array

Esistono tre diverse tipologie di array in PHP:

- Array con indici numerici: array i cui elementi sono memorizzati e accessibili in maniera lineare tramite un indice numerico.
- **Array associativi**: array i cui elementi sono individuati tramite una chiave di **tipo string**.
- **Array multidimensionali**: array i cui elementi contengono altri array, formando una o più matrici di valori.

In questa guida analizziamo gli array con indici numerici.

Inizializzare un array

Nella pratica gli array sono una serie di **variabili** tutte con lo stesso nome, ma con indice diverso.

PHP mette a disposizione due modi per inizializzare un array:

1. Tramite il costrutto array().
2. Tramite la sintassi con le parentesi quadre [], disponibile da PHP 5.4.

In entrambi i casi gli elementi vanno separati da una virgola. Vediamo un esempio di utilizzo:

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];

$carBrand = array('Ferrari', 'Porsche', 'Lamborghini');
```

```
echo "$carBrand[0]\n";

echo "$carBrand[1]\n";

echo "$carBrand[2]\n";

?>
```

Nell'esempio abbiamo stampato a video gli elementi dell'array, semplicemente accedendo a ciascuno di loro tramite l'indice numerico.

> **Nota:** per comodità è preferibile utilizzare la sintatti con le parentesi quadre.
>
> **Nota:** gli indici numerici vengono assegnati automaticamente e partono da 0. Vedremo più avanti che è possibile aggiungere elementi all'array, specificando un indice a piacimento.

Accedere ad un elemento

Oltre ad utilizzare direttamente l'indice numerico, è possibile accedere all'elemento di un array tramite alcune **funzioni PHP** speciali, che tengono traccia degli elementi dell'array tramite un puntatore:

- current() restituisce il valore dell'elemento corrente.
- key() restituisce l'indice dell'elemento corrente.
- next() restituisce l'elemento successivo, spostando di una posizione avanti il puntatore.
- prev() restituisce l'elemento precedente, spostando di una posizione indietro il puntatore.
- reset() restituisce il primo elemento, spostando il puntatore all'inizio dell'array.
- end() restituisce l'ultimo elemento, spostando il puntatore alla fine dell'array.

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];



echo "Elemento corrente ".current($carBrand).", ".key($carBrand)."\n"; // Elemento corrente Ferrari, 0

next($carBrand);

echo "Elemento corrente ".current($carBrand).", ".key($carBrand)."\n"; // Elemento corrente Porsche, 1

prev($carBrand);
```

```
echo "Elemento corrente ".current($carBrand)., ".key($carBrand)."\n"; // Elemento corrente Ferrari, 0

reset($carBrand);

echo "Elemento corrente ".current($carBrand)., ".key($carBrand)."\n"; // Elemento corrente Ferrari, 0

end($carBrand);

echo "Elemento corrente ".current($carBrand)., ".key($carBrand)."\n"; // Elemento corrente Lamborghini, 2
?>
```

Aggiungere un elemento

Per aggiungere un nuovo elemento ad un array è sufficiente utilizzare la seguente sintassi:

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];

$carBrand[] = 'Mclaren'; // Nuovo elemento


echo "$carBrand[3]"; // Output: Mclaren


$carBrand[25] = 'Bentley'; // Nuovo elemento


echo $carBrand[25]; // Output: Bentley

?>
```

Quando si aggiungono nuovi elementi ad un array e non si specifica un indice, PHP riprende la numerazione partendo dall'indice numerico più alto. Dunque un nuovo elemento inserito avrà indice pari all'indice più grande incrementato di uno.

Nel nostro esempio, prima di inserire nuovi elementi, l'indice più alto è 2, quindi l'elemento inserito alla riga 3 avrà indice pari a 3.

Alla riga 7 abbiamo invece specificato 25 come indice, dunque un eventuale nuovo elemento inserito avrebbe indice pari a 26.

Modificare un elemento

È possibile modificare gli elementi di un array semplicemente riassegnando il valore ad un elemento tramite il suo indice numerico.

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];

echo "$carBrand[1]\n"; // Porsche

$carBrand[1] = 'Bentley';

echo "$carBrand[1]"; // Bentley

?>
```

Eliminare un elemento

Esistono diversi modi per eliminare un elemento da un array, uno di questi è tramite la funzione unset(), utilizzata per eliminare qualsiasi variabile PHP dalla memoria.

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];



echo "$carBrand[1]\n"; // stampa correttamente il valore Porsche



unset($carBrand[1]); // Dopo questa istruzione gli elementi dell'array avranno indici 0, 2



echo "$carBrand[1]"; // Genera un'errore
?>
```

Nell'esempio abbiamo stampato il valore memorizzato nella variabile $carBrand[1], prima e dopo l'eliminazione dell'elemento. Nel secondo caso otteniamo il seguente errore

PHP Notice: Undefined offset: 1 in C:\Users\Zucchino\Desktop\test.php on line 8

in quanto stiamo tentando di accedere ad una posizione dell'array non più presente.

> **Nota:** la cancellazione di un elemento provoca un "buco" nell'indicizzazione dell'array. Se si aggiunge un nuovo elemento, PHP riprende la numerazione partendo dall'indice numerico più alto.

La funzione array_values()

È possibile utilizzare la funzione array_values() per riassegnare gli indici numerici a seguito di una o più cancellazioni degli elementi di un array.

Di seguito è mostrato un esempio

```
<?php

$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];



echo "$carBrand[1]\n"; // stampa correttamente il valore Porsche



unset($carBrand[1]); // Dopo questa istruzione gli elementi dell'array avranno indici 0, 2



$carBrand = array_values($carBrand);



echo "$carBrand[1]"; // stampa il valore Lamborghini

?>
```

Stampare un array

Per stampare a video gli elementi di un array possiamo utilizzare una delle seguenti funzioni PHP: print_r() oppure var_dump().

Vediamo un esempio di utilizzo per capire la differenza

```
<?php
$carBrand = ['Ferrari', 'Porsche', 'Lamborghini'];
print_r($carBrand);
var_dump($carBrand);
/*
Array(
    [0] => Ferrari
    [1] => Porsche
    [2] => Lamborghini
)
array(3) {
  [0]=>
  string(7) "Ferrari"
  [1]=>
  string(7) "Porsche"
  [2]=>
  string(11) "Lamborghini"
*/
?>
```

Nel caso di print_r() vengono stampati i valori dell'array con i corrispettivi indici.

Nel caso di var_dump() vengono stampati i valori dell'array così come sono memorizzati, con l'indice associato e il tipo di dato. La funzione var_dump() viene utilizzata per operazioni di debug del codice, ovvero per risolvere bug nel codice visualizzando lo stato delle variabili.

Array associativi

In termini di funzionalità gli array associativi sono molto simili agli **array** con indici numerici, ma sono diversi in termini di indice. Nel caso degli array associativi, gli elementi vengono individuati da un indice di tipo stringa, chiamato chiave (*key*), che stabilisce un associazione tra chiavi e valori.

Vediamo subito un esempio di utilizzo:

```
<?php

$array = [

    'nome' => 'Mario',

    'cognome' => 'Rossi',

    'eta' => '35',

    'lavoro' => 'programmatore'

];



echo $array['nome']."\n"; // Mario

echo "$array[lavoro]"; // programmatore



// l'istruzione echo "$array['lavoro']"; genera un errore di sintassi

?>
```

L'associazione tra chiavi e valori viene fatta attraverso l'operatore di assegnamento =>, mentre è possibile riferirsi ad un elemento tramite la sua chiave.

> **Nota:** quando si fa l'echo di una stringa tra doppi apici contenente l'elemento di un array associativo, è necessario riferirsi all'elemento omettendo gli apici della chiave, altrimenti il programma genera un errore di sintassi.

Utilizzo degli array associativi

Grazie alla loro potenza e flessibilità, gli array associativi sono molto utilizzati in PHP.

Un tipico utilizzo è con la funzione di PHP in_array(), che verifica la presenza di un elemento all'interno di un insieme di valori, come mostrato nell'esempio:

```
<?php

$langs = [

    'it' => 'italiano',

    'en' => 'inglese',

    'es' => 'spagnolo',

    'fr' => 'francese',

    'de' => 'tedesco',

    'ru' => 'russo'

];

if (in_array('spagnolo', $langs)) {

    echo "Lingua supportata";

} else {

    echo "Lingua non supportata";

}?>
```

Array multidimensionali

PHP consente la creazione di **array** di array, noti come **array multidimensionali**.

Sono utili quando sono necessarie due serie di indici, come ad esempio nel caso delle coordinate di una mappa o di un grafico.

Facciamo un esempio di array multidimensionale:

```
<?php

$maps = [
    "valletta" => [

        "longitudine" => "35.917973",

        "latitudine" => "14.449943"

    ],

    "londra" => [

        "longitudine" => "52.9085300",

        "latitudine" => "-3.8257400"

    ],

    "roma" => [

        "longitudine" => "41.924824",

        "latitudine" => "12.497353"

    ]

];

?>
```

Funzioni di ordinamento array

Tra le operazioni che si possono effettuare con gli **array**, non può mancare la possibilità di ordinare gli elementi in un determinato ordine.

PHP mette a disposizione diverse **funzioni** native per **ordinare in ordine crescente o decrescente gli elementi di un array**, in ordine alfabetico se gli elementi sono stringhe, dal più piccolo al più grande o viceversa se gli elementi sono numeri.

Di seguito le funzioni più comunemente usate ed esempi di utilizzo di tali funzioni:

- sort(): consente l'ordinamento di array numerici in ordine crescente.
- rsort(): consente l'ordinamento di array numerici in ordine decrescente.
- asort(): consente l'ordinamento in ordine crescente di **array associativi** in base al valore degli elementi.
- arsort(): consente l'ordinamento in ordine decrescente di array associativi in base al valore degli elementi.
- ksort(): consente l'ordinamento in ordine crescente di array associativi in base alla chiave degli elementi.
- krsort(): consente l'ordinamento in ordine decrescente di array associativi in base alla chiave degli elementi.

Ordinamento di array numerici

La funzione sort() viene utilizzata per ordinare gli elementi di un array numerico in ordine crescente:

```
<?php

$names = ["Mario", "Alberto", "Pippo", "Tizio", "Caio", "Sempronio"];

sort($names);

var_dump($names);

/* Output

array(6) {

  [0]=>

  string(7) "Alberto"

  [1]=>

  string(4) "Caio"
```

```
  [2]=>

  string(5) "Mario"

  [3]=>

  string(5) "Pippo"

  [4]=>

  string(9) "Sempronio"

  [5]=>

  string(5) "Tizio"

}




*/
?>
```

La funzione rsort() invece consente l'ordinamento degli elementi in ordine decrescente:

```
<?php

$numbers = [12, 50, 8, 43, 1, 14];

rsort($numbers);

var_dump($numbers);

/* Output
```

```
array(6) {

  [0]=>

  int(50)

  [1]=>

  int(43)

  [2]=>

  int(14)

  [3]=>

  int(12)

  [4]=>

  int(8)

  [5]=>

  int(1)

}

*/

?>
```

Nota: le associazioni tra indici ed elementi vengono perse, a seguito dell'ordinamento tramite le funzioni sort() e asort().

Ordinamento di array associativi in base al valore

La funzione asort() viene utilizzata per ordinare un array associativo in ordine crescente, in base al valore degli elementi:

```
<?php

$countries = [

    "en" => "Inghilterra",

    "de" => "Germania",

    "es" => "Spagna",

    "it" => "Italia",

    "fr" => "Francia"

];

asort($countries);

var_dump($countries);

/* Output

array(5) {

  ["fr"]=>

  string(7) "Francia"

  ["de"]=>

  string(8) "Germania"

  ["en"]=>
```

```
  string(11) "Inghilterra"

  ["it"]=>

  string(6) "Italia"

  ["es"]=>

  string(6) "Spagna"

}

*/

?>
```

Al contrario la funzione arsort() ordina gli elementi dell'array associativo in maniera decrescente, sempre in base al loro valore:

```
<?php

$colors = [

    'red' => "rosso",

    'green' => "verde",

    'yellow' => "giallo",

    'white' => "bianco",

    'black' => "nero"

];
```

```
arsort($colors);

var_dump($colors);

/* Output

array(5) {

  ["green"]=>

  string(5) "verde"

  ["red"]=>

  string(5) "rosso"

  ["black"]=>

  string(4) "nero"

  ["yellow"]=>

  string(6) "giallo"

  ["white"]=>

  string(6) "bianco"

}

*/

?>
```

Ordinamento di array associativi in base alla chiave

Concludiamo questa guida con le funzioni ksort() e krsort(), che consentono di ordinare rispettivamente in ordine crescente e decrescente gli elementi di un array associativo in base alla loro chiave:

```
<?php

$numbers = [

    "10" => "dieci",

    "40" => "quaranta",

    "50" => "cinquanta",

    "20" => "venti",

    "30" => "trenta"

];

ksort($numbers); // Ordinamento crescente in base alle chiavi

var_dump($numbers);

krsort($numbers); // Ordinamento decrescente in base alle chiavi

var_dump($numbers);

/* Output

array(5) {

  [10]=>

  string(5) "dieci"
```

```
  [20]=>

  string(5) "venti"

  [30]=>

  string(6) "trenta"

  [40]=>

  string(8) "quaranta"

  [50]=>

  string(9) "cinquanta"

}

array(5) {

  [50]=>

  string(9) "cinquanta"

  [40]=>

  string(8) "quaranta"

  [30]=>

  string(6) "trenta"

  [20]=>

  string(5) "venti"

  [10]=>
```

```
  string(5) "dieci"



}



*/
?>
```

Nota: in questo l'associazione tra indici ed elementi sono preservate, in quanto si tratta di array associativi.

Ordinamento di array multidimensionali

Tra le funzioni per l'ordinamento di array, ne esiste una per l'ordinamento di **array multidimensionali**: array_multisort().

La funzione array_multisort() consente di ordinare più array insieme o un array multidimensionale, rispetto a una o più delle sue dimensioni.

Vediamo prima un esempio di utilizzo nel caso dell'ordinamento di più array:

```
<?php

$ar1 = [10, 100, 100, 0];

$ar2 = [1, 3, 2, 4];

array_multisort($ar1, $ar2);



var_dump($ar1);

var_dump($ar2);

/* Output
array(4) {

  [0]=>

  int(0)

  [1]=>

  int(10)

  [2]=>

  int(100)

  [3]=>
```

```
  int(100)

}

array(4) {

  [0]=>

  int(4)

  [1]=>

  int(1)

  [2]=>

  int(2)

  [3]=>

  int(3)

}




*/
?>
```

Per una panoramica completa sulle funzioni PHP per l'ordinamento di array, consulta il seguente indirizzo **https://www.php.net/manual/en/array.sorting.php**.

Operatori PHP, a cosa servono?

Con gli esempi mostrati finora abbiamo introdotto **gli operatori** PHP. Gli operatori **sono** simboli **utilizzati dal linguaggio per manipolare o eseguire l'elaborazione dei dati** sui valori delle **variabili**.

Ogni linguaggio di programmazione ha i suoi operatori, alcuni di questi eseguono operazioni matematiche (come la somma o la moltiplicazione), altri operano su dati di tipo stringa, altri eseguono funzioni di tipo diverso.

Tipi di operatori PHP

Gli operatori si dividono in due categorie:

1. **Operatori unari**: necessitano di un solo operando (Es. `$i++`)
2. **Operatori binari**: necessitano di due operandi (Es. `$x = $y + 10`)

Nota: PHP consente anche l'utilizzo di un operatore ternario, il `?`, che vedremo più avanti quando parleremo dell'istruzione `if-else`

In PHP gli unici operatori unari sono quelli di **incremento/decremento**, mentre quelli binari sono:

- **Operatore di stringa**: serve a concatenare le stringhe.
- **Operatori aritmetici**: usati per manipolare dati numerici tramite le comuni operazioni aritmetiche come somma, moltiplicazione, sottrazione e divisione.
- **Operatori di assegnazione**: assegnano valori alle variabili.
- **Operatori di confronto**: confrontano i valori delle variabili restituendo un valore **booleano**.
- **Operatori logici**: operatori booleani utilizzati per concatenare due o più condizioni nelle **istruzioni condizionali**.
- **Operatori bitwise**: eseguono operazioni sui singoli bit di un numero intero.

Nella guida successiva analizzeremo un operatore molto semplice: l'**operatore di stringa**.

Operatori di incremento e decremento

Come suggerisce il nome stesso, gli **operatori di incremento e decremento** in PHP servono per incrementare e decrementare di una unità il valore di una variabile.

L'operatore di incremento è rappresentato dal doppio operatore di somma `++`, mentre quello di decremento dal doppio operatore di sottrazione `--`.

Il loro utilizzo assume un significato diverso a seconda di dove vengono posizionati, prima o dopo la variabile.

Operatore	Descrizione
`++$x`	Operatore di pre-incremento. Prima viene incrementato il valore della variabile e successivamente viene valutata la variabile.
`$x++`	Operatore di post-incremento. Prima viene valutata la variabile e successivamente viene incrementato il valore della stessa.
`--$x`	Operatore di pre-decremento. prima viene decrementato il valore della variabile e successivamente viene valutata la variabile.
`$x--`	Operatore di post-decremento. Prima viene valutata la variabile e successivamente viene decrementato il valore della stessa.

Di seguito un esempio di utilizzo:

```php
<?php

$x = 15;

echo ++$x; // 16

echo $x;   // 16


$x = 15;

echo $x++; // 15

echo $x;   // 16


$x = 15;

echo --$x; // 14

echo $x;   // 14


$x = 15;

echo $x--; // 15

echo $x;   // 14
?>
```

Gli operatori di incremento e decremento sono spesso utilizzati nei **cicli iterativi**, argomento che affronteremo tra qualche guida.

Operatore di stringa o di concatenazione

PHP mette a disposizione un operatore di stringa: l'operatore di concatenazione, rappresentato dal carattere del punto ., consente di unire tra loro una o più stringhe

```
<?php

$x = "Hello";

$y = " World!";

echo $x . $y; // Hello World!

$z = " How are you?";

echo $x . $y . $z; // Hello World! How are you?
?>
```

È possibile utilizzare l'operatore di concatenazione in congiunzione all'operatore di assegnazione uguale =. Il suo significato è: prima concatena le stringhe e poi assegna il risultato alla variabile.

Vediamo un esempio:

```
<?php

$x = "Hello ";

$x .= "World!";     // $x contiene "Hello World!"
?>
```

Parleremo degli operatori di assegnazione in una guida dedicata.

Operatori aritmetici in PHP

Gli **operatori aritmetici** sono utilizzati per eseguire operazioni aritmetiche comuni come addizione, sottrazione, moltiplicazione, divisione e modulo.

Di seguito l'elenco completo degli operatori aritmetici supportati da PHP:

Operatore	Descrizione	Esempio
+	Addizione	$x + $y
-	Sottrazione	$x - $y
*	Moltiplicazione	$x * $y
/	Divisione	$x / $y
%	Modulo	$x % $y

Alcuni esempi di codice:

```
<?php
$x = 9;
$y = 2;
$z = 4;
echo $x + $y; // 11
echo $x - $y; // 7
echo $x * $y; // 18
echo $x / $y; // 4.5
echo $x % $y; // 1
echo $x + $y * $z; // 17
echo ($x + $y) * $z; // 44



?>
```

Nota: l'ordine in cui le operazioni vengono svolte è come nella matematica. Ad esempio, l'operatore di moltiplicazione ha una precedenza più alta rispetto all'operatore di addizione. Naturalmente è possibile utilizzare le parentesi tonde per modificare questo comportamento.

Operatori di assegnazione

Gli **operatori di assegnazione** in PHP vengono utilizzati per assegnare valori alle variabili durante l'esecuzione del programma. L'operatore di assegnazione base è il simbolo dell'uguale (=).

PHP mette a disposizione anche una serie di operatori che sono una combinazione tra l'operatore di assegnazione base e gli **operatori aritmetici**, oppure l'**operatore di stringa**.

La tabella seguente mostra questo tipo di operatori:

Operatore	Descrizione	Esempio	Equivale a
=	**Operatore di assegnazione base**. Assegna i valori dagli operandi di destra all'operando di sinistra.	$x = $y	$x = $y
+=	**Operatore aggiungi e assegna**. Aggiunge l'operando di destra all'operando di sinistra e assegna il risultato all'operando di sinistra.	$x += $y	$x = $x + $y
-=	**Operatore sottrai e assegna**. Sottrae l'operando di destra dall'operando di sinistra e assegna il risultato all'operando di sinistra.	$x -= $y	$x = $x - $y
*=	**Operatore moltiplica e assegna**. Moltiplica l'operando di destra con l'operando di sinistra e assegna il risultato all'operando di sinistra.	$x *= $y	$x = $x * $y
/=	**Operatore dividi e assegna quoziente**. Divide l'operando a sinistra con l'operando di destra e assegna il risultato all'operando di sinistra.	$x /= $y	$x = $x / $y
%=	**Operatore dividi e assegna modulo**. Calcola il resto della divisione tra l'operando di sinistra e quello di destra e assegna il risultato all'operando di sinistra.	$x %= $y	$x = $x % $y
**=	**Operatore eleva a potenza e assegna.** Eleva l'operando di sinitra alla potenza dell'operando di destra e assegna il risultato all'operando di sinistra.	$x **= $y	$x = $x ** $y
.=	**Operatore concatena e assegna.** Concatena l'operando di sinistra con quello di destra e assegna il risultato all'operando di sinistra.	$x .= $y	$x = $x . $y

Il seguente programma mostra gli operatori di assegnazione all'opera:

```
<?php

$x = 30;

echo $x."\n"; // 30

$x += 30;

echo $x."\n"; // 60
```

```
$x -= 20;

echo $x."\n"; // 40

$x *= 3;

echo $x."\n"; // 120

$x /= 10;

echo $x."\n"; // 12

$x %= 5;

echo $x."\n"; // 2

$x **= 4;

echo $x."\n"; // 16

$x = 'Ciao';

$y = ' a tutti!';

$x .= $y;

echo $x // Ciao a tutti!
?>
```

La variabile `$x`, inizialmente impostata a `30`, cambia il suo valore dopo ogni operazione matematica effettuata su di essa e, conseguente assegnazione tramite il simbolo di uguale.

Operatori di confronto

Gli **operatori di confronto** vengono utilizzati per confrontare due valori in modo booleano.

Di seguito l'elenco completo degli operatori di confronto:

Operatore	Descrizione	Esempio
==	Uguale a	$x == $y
===	Uguale e dello stesso tipo di	$x === $y
!=	Non uguale a	$x != $y
<>	Non uguale a (identico a !=)	$x <> $y
!==	Non uguale e di tipo diverso di	$x !== $y
<	Minore di	$x < $y
>	Maggiore di	$x > $y
<=	Minore o uguale di	$x <= $y
>=	Maggiore o uguale di	$x >= $y
<=>	Operatore *spaceship* introdotto con PHP 7. Confronta $x e $y e restituisce 1 se il $x > $y, -1 se $x < $y, 0 se $x == $y.	$x <=> $y
??	Operatore *null coalesce* introdotto con PHP 7. Utile per assegnare valori di default, confronta $x, $y e $z e restituisce il primo valore non null, se tutti i valori risultato null restituisce null.	$x ?? $y ?? $z

```php
<?php

$x = 10;

$y = "10";

var_dump($x == $y);  // bool(true)

var_dump($x === $y); // bool(false)

var_dump($x != $y);  // bool(false)

var_dump($x !== $y); // bool(true)

var_dump($x < $y);   // bool(false)
```

```
var_dump($x > $y);   // bool(false)



var_dump($x <= $y);  // bool(true)



var_dump($x >= $y);  // bool(true)
?>
```

In questo esempio abbiamo usato la funzione var_dump() per stampare il tipo e il valore del risultato dei vari confronti.

Operatori logici in PHP

Gli **operatori logici** vengono in genere utilizzati per concatenare due o più istruzioni condizionali.

Di seguito l'elenco completo degli operatori logici:

Operatore	Descrizione	Esempio
&&	Operatore logico AND. La condizione è vera solo se entrambi gli operandi sono veri.	($x && $y)
||	Operatore logico OR. La condizione è vera se almeno uno degli operandi è vero.	($x === $y)
!	Operatore logico NOT. Inverte la logica dell'operando. Se è falso, la condizione è vera e viceversa.	(! $x)
xor	Operatore logico XOR. La condizione è vera solo se uno dei due operandi è vero.	($x xor $y)

Un esempio di utilizzo:

```
<?php



$score = 150;
if ($score > 0 && $score <= 100) {
    echo "Con $score punti hai vinto un peluche piccolo";
} elseif ($score > 100 && $score <= 200) {
    echo "Con $score punti hai vinto un peluche medio";
} elseif ($score > 200) {
    echo "Con $score punti hai vinto un peluche grande";
} else {
    echo "Spiacente non hai vinto nulla";
}



?>
```

Nota: è possibile utilizzare le sintatti and e or, ma per convenzione è preferibile usare le forme contratte && e ||.

Operatori bitwise in PHP

Gli **operatori bitwise** agiscono direttamente sui bit dei numeri.

Anche se possono risultare di non facile comprensione, conoscerli è importante per il loro stretto rapporto con la matematica binaria, che è la base dei computer.

Infatti mentre gli umani sono abituati a manipolare i numeri in base 10, il linguaggio macchina è in base 2, che comprende solo 2 valori: 0 (false) e 1 (true).

In PHP gli operatori che lavorano a livello di bit sono:

Operatore	Descrizione	Esempio
&	Operatore binario. Esegue l'AND bit a bit di due numeri. Il risultato è 1 solo se entrambi i bit sono 1.	$x & $y
|	Operatore binario. Esegue l'OR (or inclusivo) bit a bit di due numeri. Il risultato è 1 se almeno uno dei bit è 1.	$x | $y
^	Operatore binario. Esegue lo XOR (or esclusivo) bit a bit di due numeri. Il risultato è 1 solo se uno dei bit è 1, ma non tutti e due.	$x ^ $y
~	Operatore unario. Inverte i bit del numero, 1 diventa 0 e viceversa.	~ $x
<<	Operatore binario. Sposta i bit del primo numero a sinistra. Il secondo numero indica in numero di posizioni da spostare. Equivale a moltiplicare il numero per 2.	$a << $b
>>	Operatore binario. Sposta i bit del primo numero a destra. Il secondo numero indica in numero di posizioni da spostare. Equivale a dividere il numero per 2.	$a >> $b

Il seguente programma mostra gli operatori bitwise all'opera:

```
<?php
$x = 127; // 1111111 in base 2
$y = 85; // 1010101 in base 2
var_dump(decbin($x & $y)); // string(7) 1010101
var_dump(decbin($x | $y)); // string(7) 1111111
var_dump(decbin($x ^ $y)); // string(6) 101010
var_dump(decbin(~ $x)); // string(64) 1111111111111111111111111111111111111111111111111111111111110000000
var_dump(decbin($x << 1)); // string(8) 11111110
var_dump(decbin($y >> 2)); // string(5) 10101

?>
```

Nell'esempio abbiamo utilizzato la funzione decbin() che converte un numero da decimale a binario, restituendo il risultato sottoforma di stringa.

Il risultato di ogni operazione viene mostrato a video tramite la funzione var_dump(), che stampa il tipo di dato (string), la lunghezza e il valore.

Strutture di controllo: il cuore della logica di programmazione

Tipiche della **programmazione imperativa**, le **strutture di controllo** consentono di controllare il flusso di esecuzione di un programma. Generalmente questo viene eseguito in sequenza, riga per riga, e una struttura di controllo consente di modificare tale flusso in base a determinate condizioni.

Come vedremo le strutture di controllo servono a specificare *se*, *quando*, *in quale ordine* e *quante volte* devono essere eseguite determinate istruzioni.

Come per altri linguaggi di programmazione, anche in PHP le strutture di controllo sono caratteristiche fondamentali che, in base al tipo di istruzione, consentono a un programma di reagire in modo diverso in base al valore della condizione impostata (istruzioni condizionali), oppure consentono la gestione di attività ripetitive (cicli iterativi).

La tabella seguente elenca le principali strutture di controllo supportate dal linguaggio

Operatore	Descrizione
if	Le **istruzioni condizionali** consentono di eseguire blocchi di codice a seconda che una condizione sia vera o falsa.
else	
elseifO else if	
switch	
while	I **cicli iterativi** consentono di eseguire diverse volte lo stesso blocco di codice in base ad una determinata condizione.
do-while	
for	
foreach	
break	Utilizzata all'interno dei cicli iterativi per interrompere l'esecuzione del ciclo e all'interno dell'istruzione switch.
continue	Utilizzato all'interno dei cicli, consente di saltare il blocco di codice dell'iterazione corrente e passare a quella successiva (se la condizione è ancora vera).
include	Costrutti del linguaggio utilizzati per l'**inclusione di file PHP** all'interno di altri file PHP.
require	
return	Costrutto usato per restituire un valore al termine dell'esecuzione di una **funzione PHP**.

Nelle successive guide le analizzeremo nel dettaglio attraverso vari esempi di codice.

Istruzioni condizionali: if, else, elseif, switch

Come la maggior parte dei linguaggi di programmazione, anche PHP consente di scrivere codice che esegue azioni in base ai risultati di condizioni logiche o comparative.

Ciò significa che è possibile creare condizioni di test sotto forma di espressioni che valutano sia true che false e, in base a queste, eseguire determinate azioni.

È possibile farlo tramite le **istruzioni condizionali** del linguaggio.

Indice dei contenuti

1. Istruzione if
2. Istruzione else
3. Istruzione elseif
4. Operatore ternario
5. Istruzione switch
6. Sintassi alternativa

Vediamo sintassi ed esempi di utilizzo per ciascuna di esse.

Istruzione if

La più semplice istruzione condizionale PHP è l'istruzione if , che viene utilizzata per eseguire un blocco di codice solo se la condizione specificata è vera.

Di seguito un esempio che ne descrive il funzionamento:

```
<?php

$x = 10;

$y = 8;

if ($x > $y) {

    echo "$x è maggiore di $y"; // 10 è maggiore di 8

}
?>
```

L'istruzione if è seguita da una condizione racchiusa tra parentesi tonde. Le successive righe, racchiuse tra parentesi graffe, rappresentano il blocco di codice da eseguire se la condizione viene valutata come vera.

Nota: le parentesi graffe possono essere omesse nel caso il blocco di codice sia composto da una singola istruzione, ma per chiarezza è buona prassi aggiungerle sempre.

La condizione deve dare come risultato un valore booleano, dunque per essere interpretata come tale da PHP, si devono usare gli **operatori di confronto** esaminati nella guida dedicata agli operatori PHP.

Adesso aggiungiamo un'ulteriore istruzione che ci consente di gestire la situziazione nel caso in cui la condizione fosse false.

Istruzione else

È possibile migliorare il codice precedente aggiungendo un'altra istruzione che fornisce una scelta alternativa. L'istruzione if-else consente di eseguire un blocco di codice se la condizione specificata viene valutata true e un altro blocco di codice se viene valutata false.

Modificando l'esempio precedente possiamo scrivere:

```
<?php

$x = 10;

$y = 14;

if ($x > $y) {

    echo "$x è maggiore di $y";

} else {

    echo "$x è minore o uguale di $y"; // 10 è minore o uguale di 14

}
?>
```

Istruzione elseif

È possibile combinare più condizioni tramite l'istruzione elseif (o else if).

Vediamo un esempio

```
<?php

$day = 'sabato';

if ($day == 'lunedì') {

    echo "Oggi è lunedì";

} elseif ($day == 'martedì') {

    echo "Oggi è martedì";

} elseif ($day == 'mercoledì') {

    echo "Oggi è mercoledì";

} elseif ($day == 'giovedì') {

    echo "Oggi è giovedì";

} elseif ($day == 'venerdì') {

    echo "Oggi è venerdì";

} elseif ($day == 'sabato') {

    echo "Oggi è sabato";

} elseif ($day == 'domenica') {

    echo "Oggi è domenica";
```

```
} else {



    echo "Impossibile stabilire che giorno è oggi!";



}



// Output: Oggi è sabato
?>
```

Operatore ternario

L'istruzione if-else ha una sua forma contratta rappresentata dal simbolo del punto interrogativo ?, detto **operatore ternario** in quanto richiede tre operandi:

1. la condizione da verificare
2. il valore restituito se la condizione è vera
3. il valore restituito se la condizione è falsa.

Di seguito è mostrato un esempio

```
<?php

$year = 2020;

echo ($year % 4 == 0) ? "$year è un'anno bisestile" : "$year non è un'anno bisestile"; // 2020 è un'anno bisestile

/* Forma normale

if ($year % 4 == 0) {

    echo "$year è un'anno bisestile";

} else {

    echo "$year non è un'anno bisestile";

}

*/
?>
```

L'istruzione if è stata sostituita dal punto interrogativo, mentre i due punti prendono il posto dell'istruzione else.

Nella condizione abbiamo utilizzato l'operatore modulo %, tramite il quale verifichiamo che il resto della divisione tra la variabile $year e 4 sia uguale a 0. La condizione è valutata vera, dunque viene stampato 2020 è un'anno bisestile.

Se avessimo utilizzato la versione normale, avremmo scritto 5 righe di codice anzichè una, come mostrato in fondo all'esempio.

> **Nota:** l'operatore ternario consente di ridurre la scrittura di codice. Tuttavia è buona norma utilizzarlo per istruzioni non particolarmente complesse, per una questione di leggibilità del codice.

Istruzione switch

L'istruzione switch rappresenta un'alternativa più leggibile nel caso volessimo concatenare più istruzioniElseif.

L'istruzione switch verifica una variabile rispetto a una serie di casi (case) finché trova una corrispondenza, quindi viene eseguito il blocco di codice corrispondente.

Possiamo riscrivere l'esempio dell'istruzione elseif utilizzando lo switch e il costrutto case in questo modo

```
<?php

$day = 'sabato';

switch ($day) {

    case 'lunedì':

        echo "Oggi è lunedì";

        break;

    case 'martedì':

        echo "Oggi è martedì";

        break;

    case 'mercoledì':

        echo "Oggi è mercoledì";

        break;

    case 'giovedì':

        echo "Oggi è giovedì";

        break;
```

```
    case 'venerdì':

        echo "Oggi è venerdì";

        break;

    case 'sabato':

        echo "Oggi è sabato";

        break;

    case 'domenica':

        echo "Oggi è domenica";

        break;

    default:

        echo "Impossibile stabilire che giorno è oggi!";

}

// Output: Oggi è sabato
?>
```

All'interno di ogni caso è presente l'istruzione break, che indica a PHP di saltare alla fine dello switch, una volta eseguito il codice associato al primo caso true.

Alcune considerazioni da tenere a mente:

- Nella condizione vengono confrontati solo i valori e non i tipi (confronto stretto con ===).
- Lo switch verifica caso dopo caso finché non viene trovata una corrispondenza. Se non viene trovata, viene eseguita quella di default (se definita).
- Se viene omesso il break, PHP esegue anche tutti i casi successivi finché non viene trovato un break.
- Per evitare comportamenti inattesi è importante inserire sempre il break alla fine di ciascun case.

Sintassi alternativa

Per le istruzioni condizionali, PHP consente una sintassi alternativa, che consiste nel sostituire la parentesi graffa di apertura con il carattere dei due punti (:), e la parentesi di chiusura con endif nel caso dell'istruzione if.

Si può applicare in ugual modo all'istruzione switch, con la parentesi di chiusura sostituita da endswitch.

La sintassi alternativa viene utilizzata per dare maggiore chiarezza al codice, sopratutto nel caso di programmi contenenti sia codice PHP che HTML, come mostrato nell'esempio:

```
<?php

$isLogged = true;

?>

<html>

<body>

    <h1>

    <?php if ($isLogged):?>

        Utente loggato

    <?php else:?>

        Utente non loggato

    <?php endif;?>

    </h1>

</body>

</html>
```

Includere file PHP tramite include e require

Uno dei modi per rendere il codice PHP più facile da leggere e modificare, è attraverso due **strutture di controllo** che mette a disposizione il linguaggio: include e require. Tramite questi due costrutti è possibile **includere file esterni nello script corrente** ed eseguirli.

Ciò risulta particolarmente utile nella modifica del layout di un sito web. Infatti nella maggior parte dei casi, questo è suddiviso in una sezione centrale coi contenuti del sito e due sezioni comuni a tutte le pagine: un header (con logo, menù, etc...) e un footer (dove troviamo informazioni secondarie di vario tipo).

Se è necessario apportare cambiamenti, invece di modificare decine o centinaia di file, è sufficiente modificare il file incluso che contiene il footer o l'header.

Vediamo come funzionano **include** e **require** e quali sono le differenze nell'utilizzo dell'uno o dell'altro costrutto.

Indice dei contenuti

1. **include**
2. **require**
3. **Quando utilizzare include**
4. **Quanto utilizzare require**
5. **include_once e require_once**

include

Il costrutto include permette di inserire il codice contenuto in un file PHP all'interno di un altro file PHP. L'inclusione di un file produce lo stesso effetto di un copia e incolla dello script incluso, nella posizione in cui viene chiamato include.

Se si verifica un errore durante il caricamento del file, la funzione include genera **un avviso** ma lo script continua l'esecuzione.

Supponiamo di avere il menù di un sito web memorizzato nel file menu.php

```
<li><a href="/">home</a></li>

<li><a href="/about">about</a></li>

<li><a href="/blog">blog</a></li>

<li><a href="/contatti">contatti</a></li>
```

Adesso includiamo il file menu.php all'interno di un altro file, denominato home.php in cui avremo il seguente codice

```
<html>

<body>

    <ul id="menu">

        <?php include('menu.php');?> //oppure include 'menu.php';

    </ul>

</body>

</html>
```

Il risultato finale, se visualizziamo il sorgente HTML nel browser, è il seguente

```
<html>

<body>

    <ul id="menu">

        <li><a href="/">home</a></li>

        <li><a href="/about">about</a></li>

        <li><a href="/blog">blog</a></li>

        <li><a href="/contatti">contatti</a></li>

    </ul>

</body>

</html>
```

Nota: poiché include e require, così come echo, sono costrutti del linguaggio e non funzioni è possibile omettere le parentesi tonde che racchiudono il file da includere.

Se viene incluso un file che non esiste, viene mostrato un avviso simile al seguente

Warning: include(): Failed opening 'menu2.php' for inclusion (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\home.php on line 4

ma lo script prosegue la sua esecuzione senza interrompersi.

require

Il costrutto require ha lo stesso funzionamento di include, ossia quello di includere un file PHP all'interno di un altro file PHP.

Diversamente da include, se si verifica un problema nel caricamento del file con require viene generato **un errore fatale** che interrompe l'esecuzione dello script:

Fatal error: require(): Failed opening required 'menu2.php' (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\home.php on line 4

Nota: Se si vogliono includere i file di una libreria o un file di configurazione essenziali per l'esecuzione dell'applicazione, è consigliabile l'uso di require, in quanto consente di interrompere l'esecuzione del programma se il file non esiste, notificando con un errore fatale lo sviluppatore. Nel caso di include invece l'esecuzione prosegue e l'avviso viene mostrato solamente se nel **php.ini** è abilitata la visualizzazione degli errori, tramite la seguente direttiva

```
ini_set("display_errors", 1);
```

Quando utilizzare include

Il costrutto include può essere utilizzato per includere file PHP che non sono strettamente essenziali per il funzionamento del programma, in particolare può essere utilizzato per:

- Definire variabili contenenti messaggi di errore.
- Includere porzioni di template di un sito web (menù, header, footer).

Quanto utilizzare require

Il costrutto require può essere utilizzato per:

- Includere funzioni utilizzate nel programma.
- Includere librerie esterne (es. PHPMailer).
- Includere un file di configurazione.

Vediamo un esempio per il caso "Includere un file di configurazione".

Nelle applicazioni web è abbastanza usuale inserire i **dati di configurazione** in un file a parte, separandoli dal resto del codice.

Supponiamo di avere un file denominato config-db.php, che restituisce tramite il **costrutto return** un **array associativo** contenente i dati di accesso ad un **database**

```
<?php

return [

    'db_host' => 'localhost',

    'db_user' => 'user',

    'db_pass' => 'password',

    'db_name' => 'test'

];
?>
```

Come facciamo ad ottenere questi dati per utilizzarli ove necessario? Possiamo usare il costrutto require e memorizzare il valore ritornato in una **variabile**

```
<?php

$configDB = require 'config-db.php';

print_r($configDB);
?>
```

In questo modo otteniamo due grossi vantaggi:

- Possiamo modificare i dati di configurazione in un unico punto del programma.
- Possiamo includere i dati di configurazione solo dove servono, avendoli a disposizione dentro un comodo array.

include_once e require_once

L'inclusione accidentale dello stesso file per più di una volta all'interno del nostro script può causare conflitti. Per evitare questa situazione, PHP fornisce i costrutti include_once e require_once.

Questi si comportano allo stesso modo dei rispettivi include e require con una sottile differenza: se il file è già stato incluso precedentemente, non verrà inserito nuovamente.

Supponiamo di avere un file denominato functions.php contenente il seguente codice

```
<?php

function squareArea($side)

{

    return $side * $side;

}
?>
```

Lo script seguente include per due volte il file functions.php tramite require_once

```
<?php

require_once 'functions.php';

echo squareArea(4); // 16

require_once 'functions.php';

echo squareArea(4); // 16
?>
```

Se al posto di require_once avessimo utilizzato require lo script avrebbe generato il seguente errore fatale

Fatal error: Cannot redeclare squareArea() (previously declared in C:\xampp\htdocs\functions.php:3) in C:\xampp\htdocs\functions.php on line 6

Conclusioni

In questa guida abbiamo analizzato l'inclusione di file PHP all'interno di altri file attraverso include e require. Il loro utilizzo consente di organizzare meglio il codice della nostra applicazione separandolo in parti più piccole, rendendolo più pulito, leggibile e facile da manutenere.

I due costrutti fanno sostanzialmente la stessa cosa, l'unica differenza è nel caso il file incluso non esista: con include viene mostrato un avviso che non interrompe l'esecuzione dello script, con require viene generato un errore fatale che interrompe l'esecuzione.

Costrutto return in PHP, a cosa serve e come funziona

In questa guida analizziamo brevemente una delle **strutture di controllo** più semplici ma anche più utilizzate nella programmazione, il costrutto return.

Come dice il nome stesso il **costrutto return restituisce qualcosa**, nello specifico un valore che può essere di qualsiasi **tipo**: un numero **intero** o decimale, una **stringa**, un **array**, un **booleano** e persino un oggetto come vedremo quando studieremo l'OOP.

Il suo utilizzo più comune è con le **funzioni PHP**, argomento che affronteremo a breve. Per adesso ci limitiamo a dire che una funzione è un insieme di istruzioni racchiuse tra parentesi graffe che prende un input, lo elabora e restituisce un risultato.

Come funziona return

Se viene chiamato all'interno di una funzione, il costrutto return **termina immediatamente l'esecuzione della funzione** corrente e restituisce il suo argomento come valore della funzione chiamata.

L'esecuzione dello script riprende dalla riga successiva a quella della chiamata alla funzione.

```
<?php

function sum($a, $b)

{

    $s = $a + $b;

    return $s;

}



$sum = sum(4, 9); // chiamata alla funzione sum()

echo "La somma è uguale a $sum";
?>
```

Se viene chiamato a livello globale, ossia non all'interno di una funzione, verrà terminata l'esecuzione dello script corrente.

```
<?php

$a = 4;

$b = 9;

$sum = $a + $b;

return $sum;

echo "La somma è uguale a $sum"; // Questa riga non viene eseguita
?>
```

Nel caso in cui lo script contenente return sia un **file incluso** tramite i costrutti include o require, il controllo viene passato al file chiamante e il valore dato da return verrà restituito come valore della chiamata a include, quindi potrà essere memorizzato dentro una variabile.

Supponiamo di voler inviare un'email, memorizziamo in un **array associativo** le impostazioni per l'invio e lo ritorniamo con return

```
<?php

/****************/

/*

/* mail-config.php

/*

/****************/


return [

    'host' => 'localhost',
```

```
    'port' => 465,

    'protocol' => 'ssl',

    'user' => 'mariorossi@gmail.com',

    'password' => '1234567890'

];
```

Tramite il costrutto include memorizziamo l'array delle impostazioni dentro una variabile in modo tale da poterle utilizzare ove necessario

```
<?php

/**********/

/*

/* mail.php

/*

/**********/



$config = include 'mail-config.php'; // include ritorna l'array definito in mail-config.php



// Istruzioni per l'invio di un'email

?>
```

Dagli esempi di codice appena visti possiamo dedurre che:

- È possibile restituire uno e un solo valore con return.
- Se si desidera restituire più di un valore, è necessario utilizzare un array.
- Qualsiasi istruzione successiva a return verrà ignorata.

Inoltre

- È possibile non restituire nulla, semplicemente scrivendo return;
- Il tipo di dato restituito può essere integer, float, string, array, object o null.
- È possibile restituire direttamente il risultato di una condizione. Es. return 4 > 10; restituirà false.

Costrutto return e istruzioni condizionali

Un uso tipico di return è in combinazione con le **istruzioni condizionali** e/o con un **ciclo** all'interno di una funzione.

Vediamo un esempio di utilizzo

```
<?php

function find($numbers, $find)

{

    foreach ($numbers as $number) {

        if ($number === $find) {

            return true;

        }

    }

    return false;

}

$numbers = [6, 14, 10, 15, 90, 315, 16, 2];

$find = find($numbers, 15);
```

```
if ($find) {

    echo "Numero trovato";

}
?>
```

Lo script precedente chiama la funzione find() passandogli come argomenti un array di numeri e un numero da trovare. All'interno della funzione viene eseguito un ciclo tramite foreach e verificata una condizione alla riga 5. Se viene trovata una corrispondenza, l'esecuzione della funzione viene immediatamente interrotta alla riga 6, altrimenti prosegue fino alla fine del ciclo restituendo false alla riga 9.

Vediamo un altro esempio

```
<?php

function isLapYear($year)

{

    if ($year % 4 === 0) {

        return true;

    } else {

        return false;

    }

}

$year = 2000;

if (isLapYear($year)) {
```

```
    echo "Il $year è un anno bisestile";

} ?>
```

La funzione isLapYear() verifica se un anno è bisestile oppure no ritornando rispettivamente true o false. Poiché il primo return restituisce un valore terminando subito l'esecuzione della funzione, è possibile omettere l'else successivo scrivendo direttamente return false;

```
<?php

function isLapYear($year)

{

    if ($year % 4 === 0) {

        return true;

    }

    return false;

}

$year = 2000;

if (isLapYear($year)) {

    echo "Il $year è un anno bisestile";

}
?>
```

Ora che abbiamo imparato a modificare il flusso di controllo di un programma attraverso le strutture di controllo, passiamo allo studio delle funzioni che abbiamo anticipato in questa guida.

Funzioni PHP: cosa sono, sintassi ed esempi

In questa guida analizziamo le **funzioni PHP**, uno degli strumenti messi a disposizione dal linguaggio che agevolano il compito dei programmatori. Assieme agli **array**, le funzioni sono una delle caratteristiche fondamentali di PHP.

Prima di vedere sintassi ed esempi di funzioni PHP, diamo una breve definizione.

Indice dei contenuti

1. **Cos'è una funzione**
2. **Vantaggi**
3. **Creare una funzione**
4. **Chiamare una funzione**
5. **Funzioni con parametri opzionali**
6. **Funzioni ricorsive**
7. **Visibilità delle variabili**
8. **Passaggio per valore o per riferimento**

Cos'è una funzione?

In programmazione, una **funzione è un costrutto** che racchiude una sequenza di istruzioni in un unico blocco di codice, e che prende un input sotto forma di parametro, effettua delle elaborazioni sui dati e restituisce un output.

PHP mette a disposizione un gran numero di funzioni integrate, come print_r(), isset(), trim(), etc... Tuttavia è frequente la necessità di creare funzioni personalizzate (chiamate anche **funzioni definite dall'utente**) che eseguano specifici compiti.

Prima di creare la propria funzione, è buona norma consultare sempre la **documentazione PHP** per vedere se quella particolare funzione è già disponibile.

Vantaggi

Alcuni vantaggi offerti dall'utilizzo di funzioni:

1. **Le funzioni riducono la ripetizione di codice**: una funzione consente di riutilizzare un blocco di istruzioni che svolge un determinato compito, semplicemente richiamando la funzione stessa ove necessario.
2. **Le funzioni rendono il codice più semplice da mantenere**: dato che una funzione viene creata una sola volta, apportare eventuali modifiche è più semplice in quanto bisogna modificare solamente il codice all'interno della funzione.
3. **Le funzioni facilitano l'eliminazione degli errori**: quando il programma è suddiviso in funzioni, se si verifica un errore si sa esattamente quale funzione ha causato l'errore.
4. **Le funzioni possono essere riutilizzate in altre applicazioni**: poiché una funzione è isolata dal resto del programma, è possibile riutilizzarla in altre applicazioni includendo il file che contiene tale funzione.

Creare una funzione

La sintassi di una **funzione PHP personalizzata** è la seguente:

```
function name(params)

{

    statement;

    return value;

}
```

dove:

1. name è il nome della funzione preceduto dalla keyword function.
2. params sono i parametri (facoltativi) separati da virgole e racchiusi tra parentesi.
3. statement è il blocco di codice da eseguire.
4. return è l'istruzione utilizzata per **ritornare valori**.
5. value è il valore ritornato dalla funzione e può essere di qualunque **tipo**.

Nota: la restituzione di un valore è facoltativa, infatti ci sono funzioni che svolgono il loro compito senza ritornare nulla. È buona prassi però ritornare comunque un valore true o false, per indicare che la funzione ha completato con successo il proprio lavoro.

Nota: il nome di una funzione deve iniziare con una lettera o un underscore, e può continuare con lettere, numeri e underscore. Non è case sensitive.

Chiamare una funzione

Una volta definita una funzione, è possibile chiamarla scrivendone il nome seguito dalle parentesi che racchiudono eventuali valori in ingresso separati da virgole. Questi valori vengono chiamati argomenti e possono essere sia **variabili** che valori costanti

```
name(param1, param2);
```

Il valore ritornato da una funzione può essere assegnato ad una variabile.

Di seguito è mostrato un esempio di utilizzo:

```
<?php
```

```
function rectanglePerimeter($base, $height)

{

    $perimeter = ($base + $height) * 2;

    return $perimeter;

}

$perimeter = rectanglePerimeter(4, 5);

var_dump($perimeter); // int(18)
?>
```

La funzione rectanglePerimeter() calcola il perimetro di un rettangolo, prendendo in input due parametri, $base e $height, rispettivamente base e altezza. Il valore ritornato della funzione viene memorizzato in una variabile e stampato a video tramite var_dump().

> **Nota:** dare sempre alla funzione un nome "parlante" che descriva cosa fa realmente la funzione. Le funzioni possono essere chiamate prima o dopo la loro definizione.

Funzioni con parametri opzionali

È possibile creare **funzioni con parametri opzionali**, ovvero con valori di default. Si inserisce il nome del parametro, seguito da un segno di uguale e dal valore predefinito, come mostra il seguente esempio:

```
<?php

function sum($val1, $val2 = 4)

{

    return $val1 + $val2;

}

$result = sum(4); // 8
?>
```

In questo caso è possibile chiamare la funzione specificando solamente il primo parametro, in quanto il secondo $val2 ha un valore di default. Nel caso chiamassimo la funzione con due argomenti, il valore di default del parametro opzionale verrebbe ignorato.

> **Nota:** nella definizione dei parametri di una funzione, quelli opzionali vanno inseriti sempre per ultimi subito dopo l'ultimo parametro non opzionale.

Invertendo l'ordine dei parametri dell'esempio precedente otteniamo il seguente codice:

```
<?php

function sum($val2 = 4, $val1)

{

    return $val1 + $val2;

}

$result = sum(4); // 8
?>
```

che **genera l'errore a runtime**

PHP Fatal error: Uncaught ArgumentCountError: Too few arguments to function sum(), 1 passed in ... on line 7 and exactly 2 expected in ...

poiché la funzione si aspetta due argomenti anzichè uno, in quanto il primo è opzionale.

Funzioni ricorsive

Una funzione ricorsiva è una funzione che chiama se stessa più volte fino a quando non viene soddisfatta una condizione. Le funzioni ricorsive sono spesso utilizzate per risolvere calcoli matematici complessi, o per effettuare elaborazioni su array associativi o multidimensionali.

La ricorsione si presta bene per risolvere il calcolo matematico del fattoriale. Il **fattoriale di un numero** è il prodotto dei numeri interi positivi minori o uguali a tale numero e si indica tramite la seguente notazione

```
n!
```

dove n è il numero di cui si vuole calcolare il fattoriale.

Ad esempio il fattoriale di 5 viene indicato con 5! ed è uguale al prodotto di 5 * 4 * 3 * 2 * 1.

Di seguito è mostrato il codice PHP di esempio:

```
<?php

$num = 5;

function factorial($num)

{

    if ($num &amp;amp;lt;= 1) {

        return 1;

    } else {

        return $num * factorial($num - 1);

    }

}
```

```
echo "Il fattoriale di $num è ".factorial($num); // 120
?>
```

Ad ogni chiamata della funzione factorial() alla riga 9, il numero viene decrementato di una unità fino a quando non viene raggiunto il caso base (riga 6), ossia quando viene soddisfatta la condizione n <= 1 che consente di uscire dalla ricorsione.

Visibilità delle variabili

La **visibilità (o scope) di una variabile** indica il contesto in cui essa è accessibile all'interno di un programma.

Le variabili contenute all'interno di una funzione hanno visibilità locale, nel senso che possono essere utilizzate solo all'interno della funzione stessa, per questo vengono dette variabili locali, in contrapposizione a quelle globali.

Come dice il nome stesso, le variabili globali hanno una visibilità più ampia rispetto a quelle locali, infatti possono essere utilizzate in qualsiasi punto del programma.

Facciamo un esempio per capire meglio:

```
<?php
function test()
{
    $x = 'dentro';
    echo $x;
}
$x = 'fuori';
test(); // dentro
echo $x; // fuori



?>
```

Anche se hanno lo stesso nome, le due variabili $x sono in realtà differenti. La prima $x, definita all'interno della funzione, ha visibilità locale, mentre quella definita all'esterno ha visibilità globale. L'output finale sarà la stampa di due valori diversi.

> **Nota:** è buona prassi evitare l'utilizzo di variabili globali, in quanto rendono difficile l'individuazione di eventuali errori.

Passaggio per valore o per riferimento

In PHP abbiamo a disposizione due modi per **passare argomenti** a una funzione: **per valore e per riferimento**. Di default, gli argomenti della funzione vengono passati per valore, ossia i valori passati alla funzione vengono copiati nelle variabili definite nei parametri. In questo caso, all'interno della funzione non viene modificato il valore della variabile originale.

Nel passaggio per riferimento invece, non viene copiato il valore ma bensì la locazione di memoria della variabile e usata come puntatore per riferirsi ad essa. Così facendo il valore della variabile verrà modificato all'interno della funzione.

Il passaggio di una variabile per riferimento viene eseguito anteponendo una e commerciale (&) al nome della variabile, come mostrato nell'esempio seguente:

```
<?php

function myFunction(&$x)

{

    $x = 5;

    return $x;

}

$x = 10;

echo $x; // 10

myFunction($x);

echo $x; // 5
```

La variabile `$x` viene stampata a video prima e dopo la chiamata alla funzione. Il suo valore cambia, in quanto stiamo passando un riferimento ad essa.

Quando è utile passare un valore per riferimento? Se stiamo lavorando con un array e vogliamo modificarne i valori all'interno di una funzione, possiamo passare alla funzione il riferimento all'array.

A partire da PHP 5.3, il linguaggio mette a disposizione le **funzioni anonime**, di cui parleremo nella prossima guida.

Funzioni anonime in PHP: cosa sono, sintassi e casi d'uso.

A partire da PHP 5.3, è possibile **creare funzioni anonime**, conosciute anche con il nome di *closure*. Questa funzionalità di PHP non viene usata spesso, ma può tornare utile in diverse occasioni, tipo nel caso dovessimo passare una funzione come argomento di un'altra funzione.

Prima di addentrarci negli esempi pratici, vediamo innanzitutto cos'è una funzione anonima.

Indice dei contenuti

1. Cos'è una funzione anonima
2. Sintassi
3. Casi d'uso
1. Assegnare una funzione anonima a una variabile
2. Funzioni anonime e callbacks
3. Funzioni anonime e Closure

Cos'è una funzione anonima

Da Wikipedia:

In programmazione informatica, una funzione anonima o funzione lambda è una funzione definita, e possibilmente chiamata, senza essere legata ad un identificatore. Le funzioni anonime sono utili per passare come argomento una funzione di ordine superiore e si trovano in linguaggi che supportano funzioni di prima classe come Haskell.

Da questa definizione possiamo subito intuire che una funzione anonima, come dice il nome stesso, è una funzione senza un nome.

Sintassi

Nella guida precedente abbiamo visto come definire una funzione normale

```
<?php

function myFunction() {

    // codice

}
```

e come chiamarla per utilizzarla nei nostri programmi

```
<?php

myFunction();
```

Le funzioni anonime sono del tutto simili alle **funzioni** normali, in quanto possono contenere del codice, accettano argomenti e restituiscono dei valori. La differenza più evidente è nel nome, infatti una funzione anonima è una **funzione senza un nome**. Il seguente codice mostra un esempio di utilizzo:

```
<?php

function ($param1, $param2) {

    //codice

};
```

Possiamo notare due sostanziali differenze con le funzioni normali:

- La funzione non ha un nome, infatti dopo la keyword `function` ci sono direttamente le parentesi.
- Dopo la definizione della funzione, c'è un punto e virgola (;). Questo indica che le funzioni anonime sono espressioni, mentre le funzioni normali sono costrutti del linguaggio.

Casi d'uso

Abbiamo visto la sintassi delle funzioni anonime, ma non sappiamo ancora come utilizzarle all'interno dei nostri programmi. Come è possibile chiamare una funzione di cui non si conosce il nome? Vediamo tre casi di utilizzo con esempi di codice. PHP ci consente di:

1. Assegnare una funzione anonima a una **variabile**, quindi chiamare la funzione utilizzando il nome della variabile.
2. Usare una funzione anonima come funzione di **callback**, ossia come parametro di un'altra funzione.
3. Utilizzare la funzione anonima per creare **closure**.

Analizziamo i tre casi con degli esempi pratici.

Assegnare una funzione anonima a una variabile

Il caso più comune per utilizzare una funzione anonima è assegnarla a una **variabile**. La riga 6 dell'esempio seguente mostra come sia possibile eseguire la variabile `$sum` come se fosse una funzione.

```
<?php

$sum = function($val1, $val2) {

    return $val1 + $val2;

};
```

```
echo $sum(3, 4); // Output: 7
```

Nota: a differenza delle funzioni normali, qui non possiamo chiamare la funzione prima della sua dichiarazione. Se nell'esempio spostiamo l'echo sopra la variabile $sum, lo script dà errore.

Funzioni anonime e callbacks

Un altro utilizzo tipico delle funzioni anonime è per la creazione di una funzione di callback. Una **funzione di callback** è una funzione che è possibile passare a un'altra funzione come argomento.

PHP ha molte funzioni native che accettano una funzione di callback, ma è anche possibile creare la propria funzione che accetta una callback.

Un esempio tipico si ha con la funzione PHP array_map() che, preso un array e una callback come argomenti, applica la funzione di callback a ciascun elemento dell'array.

Vediamo un esempio pratico

```
<?php

$array = [2, 4, 6];

array_map(function($val) {

        echo ($val * 2)."\n";

}, $array);

/* Output

4

8

12

*/
```

La funzione di callback calcola il doppio di ciascun elemento dell'array $array passato come argomento. Il codice precedente può anche essere riscritto assegnando la funzione anonima ad una variabile, per essere utilizzata come argomento di array_map()

```
<?php

$array = [2, 4, 6];

$double = function($val) {

          echo ($val * 2)."\n";

};

array_map($double, $array);

/* Output

4

8

12

*/
```

Funzioni anonime e closure

È possibile utilizzare le funzioni anonime per la creazione di closure (letteralmente *chiusura*). Con una **closure** è possibile utilizzare variabili esterne all'interno di una funzione anonima, anche se tali variabili non sono accessibili all'interno della stessa, in quanto hanno ambito locale.

Vediamo meglio con un esempio pratico

```
<?php

function helloFunction() {

    $hello = "buongiorno";

    return function($name) use ($hello) {

        $hello = ucfirst($hello);

        return "$hello $name!";

    };

};

$helloFunction = helloFunction();

echo $helloFunction("Programmatori PHP"); // Buongiorno Programmatori PHP!
```

Analizziamo il codice passo passo:

1. La funzione helloFunction() inizializza la variabile locale $hello e restituisce una funzione anonima.
2. Nelle righe 6-9, la funzione anonima modifica la variabile $hello convertendo la prima lettera in maiuscolo e ritorna una stringa contenente i valori di $hello e $name.
   Possiamo notare la keyword use che ci consente di utillizzare la variabile esterna $hello all'interno della funzione anonima. Infatti normalmente, la funzione anonima non avrebbe accesso a questa variabile, poiché essa ha un ambito locale solamente per la funzione helloFunction().
3. Alla riga 12 chiamiamo la funzione helloFunction() e assegnamo il valore restituito (ossia quello della funzione anonima) alla variabile $helloFunction.
4. A questo punto, helloFunction() ha terminato la sua esecuzione. Normalmente, la variabile locale $hello sarebbe fuori dall'ambito della funzione anonima che viene chiamata alla riga successiva. Tuttavia, poiché abbiamo creato una closure, la funzione anonima può comunque accedere a questa variabile.
5. Alla riga 13, chiamiamo la nostra funzione anonima che ritorna la stringa di testo "Buongiorno Programmatori PHP!".

La cosa importante da notare nell'esempio, è che la funzione anonima restituita può accedere alla variabile locale della funzione normale che racchiude, anche dopo che questa ha terminato la sua esecuzione.

Ciò evita l'uso di una cattiva pratica di programmazione, ossia l'utilizzo della keyword global, che consente di dichiarare come globale una variabile locale ampliando il suo scope.

In conclusione **le funzioni anonime risultano utili nel caso in cui si abbia la necessità di utilizzare delle funzioni usa e getta** per svolgere operazioni specifiche, senza doverle quindi definire come funzioni normali.

PHP 7: dichiarazione dei tipi in ingresso e uscita delle funzioni

Una delle novità più importanti di PHP 7 è stata l'introduzione della **dichiarazione dei tipi in ingresso e in uscita delle funzioni**. In particolare a partire da PHP 7, per i parametri in ingresso è possibile dichiarare valori di tipo scalare: **int**, **float**, **string** e **bool**.

Questi vanno ad aggiungersi al tipo class e al **tipo array**, disponibili rispettivamente da PHP 5.0 e PHP 5.1 e al tipo object che sarà aggiunto successivamente in PHP 7.2.

Oltre ai tipi scalari in ingresso, PHP 7 ha introdotto la possibilità di dichiarare anche il **tipo di dato** per i **valori restituiti dalle funzioni**.

La **dichiarazione dei tipi** (o type hinting) delle funzioni "forza" il programmatore a passare come parametri solamente valori di un determinato tipo. Questo consente di scrivere codice più robusto e facile da leggere.

Ma vediamo i benefici apportati dal type hinting attraverso alcuni esempi di codice e come poter sfruttare questa caratteristica del linguaggio.

Indice dei contenuti

1. **Cos'è il type hinting**
2. **Vantaggi**
3. **Costrutto declare**
4. **Tipi in ingresso**
5. **Tipi in uscita**

Cos'è il type hinting

Poiché PHP è un **linguaggio a tipizzazione debole**, non è necessario specificare un tipo di dati per le variabili o per i parametri delle funzioni.

Tuttavia dalla versione 5, il linguaggio consente di specificare il tipo di dati che viene passato come parametro delle funzioni. Questa funzionalità è chiamata **type hinting** o dichiarazione del tipo di dati. Se ad esempio viene passato un valore con un tipo di dato diverso da quello definito nella dichiarazione della funzione, viene generato un errore fatale.

Prima di dare uno sguardo agli esempi pratici vediamo alcuni dei vantaggi apportati dal type hinting.

Vantaggi

Il type hinting porta con sè diversi benefici tra cui:

- Evita la scrittura di codice aggiuntivo per identificare il tipo di dati
- Se si lavora in team, si riduce la possibilità che altri sviluppatori utilizzino la funzione in modo sbagliato.
- Quando si lavora con un IDE vengono dati suggerimenti sul tipo di dato accettato dalle funzioni, evitando così di commettere errori.

Vediamo come sfruttare questa funzionalità introducendo un nuovo costrutto del linguaggio.

Costrutto declare

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 **strutture di controllo**

declare (directive)

Tra le direttive che è possibile impostare c'è strict_types, che in pratica fa si che venga sollevata un eccezione (un errore) qualora non vengano rispettati i tipi dichiarati in ingresso e/o in uscita delle funzioni.

Per attivare il controllo è sufficiente inserire la seguente riga di codice all'inizio di ogni script

```
declare(strict_types=1);
```

Per disabilitare il controllo sui tipi è sufficiente rimuovere la riga precedente oppure passare come valore 0 anziché 1.

Tipi in ingresso

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('Flavio');
```

La funzione printHello() stampa a video una stringa contenente il nome passato come parametro. In questo caso abbiamo indicato string come tipo di dato del parametro $name.

Vediamo cosa succede se passiamo un valore di tipo int

```php
<?php

declare(strict_types=1);

function printHello(string $name)

{

    echo "Hello $name";

}

printHello(4); // genera un errore
```

Se eseguiamo il codice precedente otteniamo il seguente errore

```
PHP Fatal error:  Uncaught TypeError: Argument 1 passed to printHello() must be of the type string, int given...
```

L'**interprete PHP** solleva un eccezione di tipo TypeError poiché non stiamo passando il tipo di dato corretto alla funzione, la quale si aspetta una stringa piuttosto che un intero.

> **Nota:** se impostiamo lo strict_types a 0, il programma viene correttamente eseguito senza generare errori.

Tipi in uscita

PHP 7 offre anche la possibilità di **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, seguito dal nome del tipo che vogliamo restituire

Vediamo un esempio di utilizzo

```php
<?php

declare(strict_types=1);

function sum(int $a, int $b): int

{
```

```
    return $a + $b;

}

echo sum(4, 9);
```

La funzione deve restituire un valore di tipo int, altrimenti viene generato un errore. Infatti se modifichiamo il codice precedente in questo modo

```
<?php

declare(strict_types=1);

function sum(int $a, int $b): int

{

    return 4.4 + $b;

}

echo sum(4, 9);
```

e lo eseguiamo, otteniamo il seguente errore

PHP Fatal error: Uncaught TypeError: Return value of sum() must be of the type int, float returned...

poiché il valore restituito dalla funzione sum() sarà di tipo float.

> **Nota:** a partire da PHP 7.1 è possibile utilizzare come risultato di una funzione il valore void, che indica che la funzione non restituisce alcun valore. Se la funzione restituisce un valore viene generato un errore.

```
<?php

declare(strict_types=1);
```

```
function printHello(string $name): void

{

    return "Hello $name";

}


printHello('Flavio');
```

Il codice genera il seguente errore poiché stiamo ritornando un valore nonostante il tipo in uscita impostato a void

PHP Fatal error: A void function must not return a value...

Funzioni PHP per gestire array

PHP mette a disposizione numerose **funzioni** per lavorare con gli **array**. Non è necessario installare estensioni per poterle utilizzare, in quanto le funzioni sono già integrate nel core del linguaggio.

Di seguito alcune funzioni per creare, manipolare, filtrare, confrontare e ordinare array. Per la lista completa **consulta la documentazione**.

Indice dei contenuti

1. array_change_key_case
2. array_chunk
3. array_column
4. array_combine
5. array_count_values
6. array_diff
7. array_fill
8. array_filter
9. array_flip
10. array_intersect
11. array_key_exists
12. array_map
13. array_merge
14. array_pop
15. array_push
16. array_search
17. array_sum
18. array
19. count
20. in_array
21. Funzioni per accedere agli elementi
22. Funzioni di ordinamento

array_change_key_case

array_change_key_case (array $array [, int $case = CASE_LOWER]) : array

La funzione array_change_key_case() ritorna una array con le chiavi convertite in minuscolo o maiuscolo. Se l'array ha indici numerici, questi non vengono convertiti.

Oltre all'array di partenza, la funzione prende una costante come parametro che di default è CASE_LOWER, per la conversione in minuscolo. L'altra costante è CASE_UPPER, per la conversione in maiuscole.

```
<?php

$colors = ['rosso' => 'red', 'giallo' => 'yellow'];

print_r(array_change_key_case($colors, CASE_UPPER));
```

```
/* Output

Array

(

    [ROSSO] => red

    [GIALLO] => yellow

)

*/
```

array_chunk

array_chunk (array $array , int $size [, bool $preserve_keys = FALSE]) : array

La funzione array_chunk() divide un array in array di dimensione uguale al parametro $size. L'ultimo array può contenere meno di elementi di $size. Con il parametro $preserve_key impostato a true, le chiavi vengono mantenute, altrimenti vengono reindicizzate nei vari array.

```
<?php

$array = ['a', 'b', 'c', 'd', 'e'];

print_r(array_chunk($array, 2));

/* Output

Array

(

    [0] => Array

        (
```

```
            [0] => a

            [1] => b

        )



    [1] => Array

        (

            [0] => c

            [1] => d

        )



    [2] => Array

        (

            [0] => e

        )



)

*/
```

array_column

array_column (array $input , mixed $column_key [, mixed $index_key = NULL]) : array

La funzione array_column() viene utilizzata con gli array multidimensionali e ritorna un array coi valori presi da una singola chiave dell'input, identificata da $column_key.

L'array ritornato viene reindicizzato numericamente, ma è possibile specificare una chiave dell'array di partenza, fornita tramite il parametro $index_key.

```
<?php

// $list rappresenta un possibile array di utenti presi da un database

$list = [

    [

        'id' => 2135,

        'name' => 'John',

        'surname' => 'Doe',

    ],

    [

        'id' => 3245,

        'name' => 'Sally',

        'surname' => 'Smith',

    ],

    [

        'id' => 5342,

        'name' => 'Jane',
```

```
        'surname' => 'Jones',

    ]

];

 $name = array_column($list, 'name');

print_r($name);

$name = array_column($list, 'name', 'id');

print_r($name);

/* Output

Array

(

    [0] => John

    [1] => Sally

    [2] => Jane

)

Array

(

    [2135] => John

    [3245] => Sally

    [5342] => Jane
```

```
)

*/
```

array_combine

array_combine (array $keys , array $values) : array

La funzione array_combine() restituisce un array creato utilizzando come chiavi, i valori dell'array $keys e, come valori, i valori dell'array $values.

```
<?php

$colors = ['verde', 'rosso', 'giallo'];

$fruits = ['avocado', 'mela', 'banana'];

$array = array_combine($colors, $fruits);

print_r($array);

/* Output

Array

(

    [verde] => avocado

    [rosso] => mela

    [giallo] => banana

)

*/
```

array_count_values

array_count_values (array $array) : array

La funzione array_count_values() ritorna un array che ha come chiavi i valori dell'array di input, e come valori, il numero di ripetizioni dei vecchi valori.

```
<?php

$array = [1, 'hello', 1, 'world', 'hello'];

print_r(array_count_values($array));

/* Output

Array

(

    [1] => 2

    [hello] => 2

    [world] => 1

)

*/
```

array_diff

array_diff (array $array1 , array $array2 [, array $...]) : array

La funzione array_diff() confronta $array1 con uno o più array e ritorna i valori di $array1 che non sono presenti in nessuno degli altri array.

```
<?php

$array1 = ["Paolo", "Andrea", "Flavio"];

$array2 = ["Andrea", "Agnese", "Paola", "Alessandra"];

$result = array_diff($array1, $array2);

print_r($result);

/* Output

Array

(

    [0] => Paolo

    [2] => Flavio

)

*/
```

array_fill

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

La funzione array_fill() crea un array di $num elementi con valore uguale al parametro $value, e le cui chiavi cominciano dal parametro $start_index.

```
<?php

$a = array_fill(2, 4, 'html');

print_r($a);

/* Output

Array

(

    [2] => html

    [3] => html

    [4] => html

    [5] => html

)

*/
```

array_filter

array_filter (array $array [, callable $callback [, int $flag = 0]]) : array

La funzione array_filter() filtra gli elementi di un array usando la funzione $callback. Le chiavi dell'array vengono mantenute.

```
<?php

// Calcolo anno bisestile

function lapYear($array)

{

    if($array % 4 == 0) {

       return true;

    }

    return false;

$array = [2014, 2015, 2016, 2020, 2010, 2008];

print_r(array_filter($array, "lapYear"));

/* Output

Array

(

    [2] => 2016

    [3] => 2020

    [5] => 2008
```

```
)

*/
```

array_flip

array_flip (array $array) : array

La funzione array_flip() inverte chiavi e valori dell'array di input e ritorna un array.

I valori dell'array di input devono essere di **tipo intero** o **stringa**. Se viene specificato un tipo diverso, la funzione genera un avviso e quel valore non verrà incluso nell'array risultante.

Se un valore compare più volte nell'array iniziale, solamente l'ultimo verrà convertito in chiave nel nuovo array, tutti gli altri andranno persi.

```
<?php

$array = ["pere", "mele", "banane"];

$flipped = array_flip($array);

print_r($flipped);

/* Output

Array

(

    [pere] => 0

    [mele] => 1

    [banane] => 2

)
```

```
*/
```

array_intersect

array_intersect (array $array1 , array $array2 [, array $...]) : array

La funzione array_intersect() confronta $array1 con uno o più array e ritorna i valori di $array1 che sono presenti negli altri array.

```
<?php

$array1 = ["Paolo", "Andrea", "Flavio"];

$array2 = ["Andrea", "Agnese", "Paola", "Alessandra"];

$result = array_intersect($array1, $array2);

print_r($result);

/* Output

Array

(

    [1] => Andrea

)

*/
```

array_key_exists

array_key_exists (mixed $key , array $array) : bool

La funzione array_key_exists() ritorna true se la chiave $key viene trovata nell'array $array, false altrimenti.

```
<?php

$array = ['lun' => 'Lunedì', 'mar' => 'Martedì', 'ven' => 'Venerdì', 'sab' => 'Sabato'];

if (array_key_exists('lun', $array)) {

    echo "Trovata";

} else {

    echo "Non trovata";

}

// Output: Trovata
```

array_map

array_map (callable $callback , array $array1 [, array $...]) : array

La funzione array_map() ritorna un array contenente il risultato dell'applicazione della funzione $callback all'array $array1 (ed altri array eventualmente forniti) usato come argomento per chiamare la funzione. Il numero di parametri accettati dalla funzione di callback deve corrispondere al numero di array passati ad array_map().

```
<?php

function cube($n)

{
```

```
    return ($n * $n * $n);

}

$a = [1, 2, 3, 4, 5];

$b = array_map('cube', $a);

print_r($b);

/* Output

Array

(

    [0] => 1

    [1] => 8

    [2] => 27

    [3] => 64

    [4] => 125

)

*/
```

array_merge

array_merge ([array $...]) : array

La funzione array_merge() ritorna un array composto dall'unione degli elementi di uno o più array presi in input.

Se gli array di input hanno chiavi di tipo stringa uguali, il valore successivo per quella chiave sovrascriverà quello precedente. Se invece gli array contengono chiavi numeriche, il valore successivo non sovrascriverà quello precedente, ma verrà aggiunto.

I valori degli array di input con chiavi numeriche, saranno reindicizzati nell'array di output partendo da zero.

```
<?php
$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
$result = array_merge($array1, $array2);
print_r($result);



/* Output



Array



(



    [color] => green



    [0] => 2



    [1] => 4



    [2] => a



    [3] => b



    [shape] => trapezoid



    [4] => 4



)



*/
```

array_pop

array_pop (array &$array) : mixed

La funzione array_pop() rimuove e restituisce il valore dell'ultimo elemento dell'array.

```
<?php

$stack = ["arancia", "banana", "mela", "lampone"];

$fruit = array_pop($stack);

print_r($stack);

/* Output

Array

(

    [0] => arancia

    [1] => banana

    [2] => mela

)

*/
```

array_push

array_push (array &$array [, mixed $...]) : int

La funzione array_push()tratta l'array come una **pila** e inserisce i nuovi valori alla fine dell'array. La lunghezza dell'array aumenta in base al numero di valori inseriti. È equivalente all'istruzione seguente:

```
<?php

$array[] = $var;
```

Di seguito un esempio di utilizzo di array_push()

```
<?php
$stack = ['orange', 'banana'];
array_push($stack, 'apple', 'raspberry');

print_r($stack);

/* Output

Array

(

    [0] => orange

    [1] => banana

    [2] => apple

    [3] => raspberry

)

*/
```

array_search

array_search (mixed $needle , array $haystack [, bool $strict = FALSE]) : mixed

La funzione array_search() effettua la ricerca del valore del parametro $needle nell'array $haystack, ritornando la prima chiave corrispondente se la ricerca ha esito positivo, false altrimenti.

Se il parametro $strict è impostato su true, la funzione array_search() farà un confronto "stretto" nell'array. Ciò significa che verranno confrontati sia il valore che il tipo di dato.

> **Nota:** Questa funzione può restituire il valore booleano false, ma può anche restituire un valore non booleano interpretato da PHP come false (ad esempio 0). Perciò è necessario utilizzare l'**operatore di confronto** === per verificare il valore restituito da questa funzione.

array_sum

array_sum (array $array) : number

La funzione array_sum() calcola la somma degli elementi di un array e ritorna il risultato.

```
<?php

$a = [2, 4, 6, 8];

echo "sum(a) = " . array_sum($a) . "\n"; // Output: 20

$b = ["a" => 1.2, "b" => 2.3, "c" => 3.4];

echo "sum(b) = " . array_sum($b) . "\n"; // Output: 6.9
```

array

array ([mixed $...]) : array

La funzione array() crea un array di elementi. Gli elementi possono essere di qualunque **tipo**. Da PHP 5.4, per creare array è preferibile utilizzare la sintassi con le parentesi quadre ([]).

Di seguito le regole di sintassi per gli elementi:

- La sintassi degli elementi è "indice => valori", separati da virgole, che definiscono indici e valori.
- L'indice può essere sia di tipo stringa che intero.
- Quando l'indice viene omesso, un indice di tipo intero viene generato automaticamente, partendo da 0.
- Se l'indice è un numero intero, l'indice generato successivo sarà l'indice intero più grande incrementato di 1.
- Quando vengono definiti due indici identici, l'ultimo sovrascrive il primo. È possibile inserire una virgola finale dopo l'ultimo elemento dell'array.

```
<?php

$array = array (

    "frutti"  => array("a" => "orange", "b" => "banana", "c" => "mela"),

    "numeri" => array(1, 2, 3, 4, 5, 6)

);
```

count

count (mixed $array_or_countable [, int $mode = COUNT_NORMAL]) : int

La funzione count() calcola la dimensione di un array, ritornando un valore di tipo intero.

Se il parametro opzionale è impostato su COUNT_RECURSIVE (o 1), la funzione conterà l'array ricorsivamente. Ciò è particolarmente utile per contare tutti gli elementi di un array multidimensionale.

```
<?php

$food = [

    'fruits' => ['orange', 'banana', 'apple'],
```

```
    'veggie' => ['carrot', 'collard', 'pea']

];

// conteggio ricorsivo

echo count($food, COUNT_RECURSIVE); // Output: 8

// conteggio normale

echo count($food); // Output: 2
```

in_array

in_array (mixed $needle , array $haystack [, bool $strict = FALSE]) : bool

La funzione in_array() ritorna true se il valore di $needle viene trovata nell'array $haystack, false altrimenti. Se $needle è una stringa, il confronto viene fatto in maniera case-sensitive.

Se il parametro opzionale $strict è impostato su true, la funzione fa un controllo anche per il tipo di dato.

```
<?php

$os = ["Mac", "Microsoft", "Solaris", "Linux"];

if (in_array("Linux", $os)) {

    echo "Sistema operativo in uso: Linux";

}

if (in_array("mac", $os)) {

    echo "Sistema operativo in uso: mac";

}
```

Funzioni per accedere agli elementi

Le funzioni current(), end(), key(), netx(), prev(), reset() consentono l'accesso agli elementi di un array. Per approfondire consulta il paragrafo relativo a **come accedere agli elementi**, nella guida dedicata agli array.

Funzioni di ordinamento

Le funzioni arsort(), asort(), krsort(), ksort(), natsort(), sort(), rsort() sono state analizzate nella guida delle **funzioni di ordinamento array**.

Funzioni PHP per gestire le stringhe

Durante lo sviluppo di un'applicazione web, capita spesso di dover **manipolare stringhe di testo** in base alle proprie esigenze.

Fortunatamente PHP mette a disposizione numerose **funzioni** per gestire le stringhe.

Vediamo quali sono quelle più utilizzate e a cosa serve ciascuna di esse.

Indice dei contenuti

1. **Calcolare la lunghezza di una stringa**
2. **Confrontare stringhe**
3. **Rimuovere spazi bianchi**
4. **Cercare una parola all'interno di una stringa**
5. **Estrarre parte di una stringa**
6. **Convertire i caratteri in maiuscolo o minuscolo**
7. **Convertire il primo carattere in maiuscolo o minuscolo**
8. **Sostituire le occorrenze di una stringa**
9. **Suddividere una stringa**
10. **Convertire gli a capo in interruzioni di riga**

Calcolare la lunghezza di una stringa

Una delle funzioni più utilizzate con le stringhe è sicuramente la funzione strlen(), che prende come argomento una stringa e ne **calcola la lunghezza in caratteri** (spazi inclusi).

Questa funzione torna utile quando vogliamo validare un valore inserito dall'utente che deve avere lunghezza fissa, come il CAP di una città.

```
<?php

$cap = "97100";

if (strlen($cap) != 5) {

    echo "Il CAP deve avere lunghezza pari a 5";

}
```

Alcune funzioni di stringa hanno anche la controparte mb_* per operare su stringhe in formato multibyte, ossia stringhe in formato Unicode come ad esempio UTF-8. Queste funzioni sono utilizzabili tramite l'estensione **mbstring** (Multibyte String) del linguaggio.

Ad esempio, la funzione strlen() ha la sua controparte nella funzione mb_strlen().

Vediamo un esempio

```
<?php

$title = "Ultime novità di PHP";

echo strlen($title)."\n"; // 21

echo mb_strlen($title)."\n"; // 20
```

La variabile $title contiene una stringa con un carattere multibyte (la a accentata à), ossia un carattere che PHP memorizza con due byte anziché uno. Ricordiamo che una stringa in PHP è memorizzata come fosse un **array** di caratteri, ciascuno dei quali occupa un byte.

Alla riga 4 l'output di strlen() dà come risultato 21, nonostante la stringa abbia lunghezza 20. Questo perché la funzione conta i byte, così come sono memorizzati nella variabile.

La funzione mb_strlen() invece, introdotta proprio per risolvere questo problema, ritorna il conteggio esatto.

Confrontare stringhe

Il **confronto tra stringhe** in PHP può essere fatto tramite l'operatore di confronto ==, che verifica l'uguaglianza tra due variabili.

Vediamo un esempio

```
<?php

$string = "Pippo";

if ($string == "Pippo") {

    echo "Le stringhe sono uguali";

} else {
```

```
    echo "Le stringhe non sono uguali";

}

// Le stringhe sono uguali
```

Tuttavia gli **operatori di confronto** non sono l'unico modo per confrontare due stringhe. PHP mette a disposizione la funzione strcmp() che prese due stringhe come parametro, ritorna un numero intero:

- minore di 0 se la prima stringa è minore della seconda.
- maggiore di 0 se la prima stringa è maggiore della seconda.
- uguale a 0 se le stringhe sono uguali.

Nota: Una stringa è minore di un'altra se viene prima di questa in ordine alfabetico. Nel caso di stringhe contenenti maiuscole, minuscole e cifre viene rispettato il seguente ordine: prima le cifre, poi le lettere maiuscole e infine le minuscole.

```
<?php

$string1 = "PHP";

$string2 = "cPanel";

$string3 = "404";

echo strcmp($string1, $string2)."\n"; // stampa -1 perché il carattere "P" viene prima di "c"

echo strcmp($string1, $string3)."\n"; // stampa 1 perché "P" viene dopo il "4"

echo strcmp($string3, $string2)."\n"; // stampa -1 perché il "4" viene prima di "c"
```

Rimuovere spazi bianchi

Quando un sito web riceve dati proveniente da un modulo contatti, può capitare che l'utente aggiunga inavvertitamente degli spazi iniziali o finali andando a "sporcare" i dati che verranno salvati in un **database**.

PHP ci viene in aiuto con tre funzioni per **pulire le stringhe da eventuali spazi**:

- ltrim() rimuove gli spazi all'inizio di una stringa.
- rtrim() rimuove gli spazi alla fine di una stringa.
- trim() rimuove gli spazi all'inizio e alla fine.

Vediamo un esempio per ciascuna delle tre funzioni

```
<?php

$name = " Mario";

$surname = "Rossi  ";

$fullname = " Mario Rossi  ";

echo ltrim($name)."\n"; // Mario

echo rtrim($surname)."\n"; // Rossi

echo trim($fullname)."\n"; // Mario Rossi
```

Nota: è consigliabile utilizzare sempre trim(), poiché se la stringa proviene da un input utente, non possiamo sapere se gli spazi siano all'inizio o alla fine.

Cercare una parola all'interno di una stringa

Capita spesso di voler **verificare se una stringa contiene una specifica parola**. In questi casi PHP mette a disposizione la funzione strpos() che restituisce la posizione della prima occorrenza della parola cercata, altrimenti false.

Vediamo la sua sintassi

```
strpos($string, $find, $offset)
```

dove:

- $string è la stringa su cui effettuare la ricerca
- $find è la parola da ricercare
- $offset è opzionale e indica la posizione da cui si vuole iniziare la ricerca.

```
<?php

$string = "Programmatore PHP freelance";

var_dump(strpos($string, 'PHP')); // int(14)

var_dump(strpos($string, 'HTML')); // bool(false)
```

Nota: la funzione strpos() può anche restituire il valore 0 se la parola cercata viene trovata all'inizio della stringa. Lo 0 è interpretato da PHP come false, dunque nella condizione per verificare la presenza della parola nella stringa è necessario utilizzare l'operatore di confronto ===, poiché oltre al valore, confronta anche il tipo di dato.

Il seguente esempio chiarisce meglio quanto appena detto

```
<?php

$find = "Buongiorno";

$string = "Buongiorno mondo";

if (strpos($string, $find) != false) { // Il confronto 0 != false è uguale a false

    echo "Trovato";
```

```
}

if (strpos($string, $find) !== false) { // Il confronto eseguito tramite !== restituisce il risultato corretto.

    echo "Trovato";

}
```

Il primo if viene interpretato false perché PHP converte il valore intero 0 in false, dunque la condizione risulta false.

Nel confronto dei tipi del secondo if, 0 come intero risulta essere diverso da false booleano, quindi la condizione risulta vera e viene correttamente stampato "Trovato".

La controparte multibyte di strpos() è mb_strpos().

Estrarre parte di una stringa

La funzione substr() consente di **estrarre una porzione di stringa**, partendo da una specifica posizione.

Sintassi di substr()

```
substr($string, $start, $lenght);
```

dove:

- $string è la stringa iniziale
- $start è la posizione da cui partire per estrarre la sottostringa. Un valore negativo indica che bisogna partire dalla fine della stringa.
- $lenght è opzionale e indica quanti caratteri bisogna estrarre partendo da $start. Un valore negativo esclude $lenght caratteri a partire dalla fine.

Vediamo un esempio per questa funzione

```
<?php

$str = "Hello World";

echo substr($str, 0, 5); // Hello

echo substr($str, 5, 6); // World

echo substr($str, -6); // World
```

La controparte multibyte di substr() è mb_substr().

Convertire i caratteri in maiuscolo o minuscolo

Le funzioni strtoupper() e strtolower() **convertono i caratteri di una data stringa rispettivamente in maiuscolo e minuscolo**.

In particolare, strtolower() torna utile per uniformare dati provenienti da un modulo contatti. Un caso abbastanza comune è un sito web che raccoglie le email degli utenti registrati, non è insolito infatti trovare alcuni indirizzi in maiscuolo e altri in minuscolo.

Di seguito un esempio di codice per entrambe le funzioni

```
<?php

$name = "Mario";

$email = "MARIO.ROSSI@GMAIL.com";

echo strtoupper($name)."\n"; // MARIO

echo strtolower($email)."\n"; // mario.rossi@gmail.com
```

Convertire il primo carattere in maiuscolo o minuscolo

Se vogliamo **convertire solamente il primo carattere di una stringa** in maiuscolo oppure in minuscolo, possiamo utilizzare rispettivamente le funzioni PHP ucfirst() e lcfirst().

In particolare, ucfirst() usato in combinazione con strtolower() si rivela utile per "aggiustare" nomi e cognomi prelevati da un **modulo registrazione utente**, prima della memorizzazione in un database.

Qui vediamo un esempio di utilizzo su variabili

```
<?php

$name = "MARIO";

$surname = "ROSSI";
```

```
echo ucfirst(strtolower($name))."\n"; // Mario

echo ucfirst(strtolower($surname))."\n"; // Rossi
```

Sostituire le occorrenze di una stringa

Per **sostituire tutte le occorrenze di una specifica stringa** all'interno di un'altra stringa, possiamo utilizzare la funzione str_replace().

La sua sintassi è la seguente

```
str_replace($search, $replace, $subject, $count)
```

dove

- $search è il valore da sostituire all'interno della stringa. Può essere un array.
- $replace è il nuovo valore che sostituirà il vecchio. Può essere un array.
- $subject è la stringa di partenza
- $count è opzionale e contiene il numero di occorrenze sostituite.

Un esempio di utilizzo di str_replace() è il seguente

```
<?php

$menu = '<ul><li><a href="/">Home</a></li><li><a href="/about">About</a></li><li><a
href="/contatti">Contatti</a></li></ul>';

$output = str_replace('<li>', '<li class="strong-text">', $menu, $count);

echo $count; // 3
```

Al termine del programma, l'argomento $count passato alla funzione str_replace() conterrà il valore 3, ossia il numero di sostituzioni effettuate.

Suddividere una stringa

Un'ulteriore funzione utilizzata spesso con le stringhe è la funzione explode(), che **suddivide una stringa** in più parti in base ad un carattere separatore e li inserisce dentro un array.

Sintassi di explode()

```
explode($delimiter, $string, $limit);
```

dove

- $delimiter è il carattere separatore.
- $string è la stringa di input.
- $limit è opzionale e stabilisce il numero massimo di parti da estrarre.

Un esempio di utilizzo

```
<?php

$string = "PHP;HTML;CSS";
$parts = explode(";", $string);
print_r($parts);

/* Output

Array

(

    [0] => "PHP"

    [1] => "HTML"

    [2] => "CSS"

)

*/
```

Convertire gli a capo in interruzioni di riga

Esiste una funzione PHP che torna utile nel caso dovessimo stampare a video una stringa proveniente da un'area di testo compilata dall'utente.

Infatti se mostriamo nel browser tale stringa, questa verrà stampata su una sola riga anche se l'utente ha digitato degli a capo (tasto Invio).

Possiamo **convertire gli a capo in interruzione di riga** (tag HTML <br />) tramite la funzione nl2br(). Vediamo subito un esempio

```
<?php

$string = "Questa è una riga di testo\nQuesto testo è su un altra riga";

echo $string."<br>";

echo nl2br($string);
```

Se eseguiamo il programma precedente tramite l'interprete, il primo echo stampa correttamente l'a capo in quanto interpreta correttamente la sequenza di escape \n.

Mentre se visualizziamo il programma con un browser, noteremo che solamente il secondo echo stampa su righe separate il contenuto della variabile $string. Questo perché nel linguaggio HTML, per andare a capo viene utilizzato il tag <br />, dunque abbiamo bisogno della funzione nl2br() per convertire la sequenza di escape \n nel corrispondente tag HTML.

Conclusioni

In questa guida abbiamo presentato alcune funzioni PHP per manipolare le stringhe. Per una consultazione completa è possibile riferirsi alla documentazione online diponibile all'indirizzo **https://www.php.net/manual/en/ref.strings.php**.

Variabili superglobali PHP: cosa sono e come utilizzarle

PHP è un linguaggio utilizzato in coppia con HTML per sviluppare applicazioni web e fornire un'interazione tra l'utente e l'applicazione stessa. Tra le possibili interazioni vi è la possibilità di inviare dati attraverso il modulo (form) HTML di un sito.

In questa guida introduciamo le **variabili superglobali**, ossia **variabili** speciali del linguaggio che consentono di gestire i dati inviati tramite un form e di recuperare informazioni sul browser (client) o il sito (server).

Per utilizzare al meglio questi strumenti è necessaria una buona comprensione di ciò che avviene tra il client e il server. Introduciamo quindi alcuni concetti come il protocollo HTTP e il suo funzionamento attraverso una richiesta e la successiva risposta.

Indice dei contenuti

1. **Il protocollo HTTP**
2. **Variabili superglobali**
3. **$_SERVER**
4. **$_GET**
5. **$_POST**
6. **GET VS POST**
7. **$_REQUEST**

Il protocollo HTTP

HTTP è il protocollo utilizzato per le comunicazioni su Internet. Consente a un client e un server di scambiarsi informazioni attraverso la rete.

Quando si visualizza la pagina di un sito, il **protocollo HTTP** consente di effettuare la richiesta al sito e di ricevere la risposta contenente il contenuto della pagina.

La richiesta HTTP contiene una serie di informazioni che saranno interpretate dal server per preparare la risposta. Di seguito è mostrato un esempio di richiesta verso la home page di questo sito

```
GET / HTTP/1.1

Accept: */*

Accept-Encoding: gzip, deflate, br

Connection: keep-alive

Host: guidaphp.it

User-agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64)
```

La prima riga di una richiesta HTTP contiene:

- il metodo HTTP (GET)
- il percorso della risorsa richiesta (/)
- la versione del protocollo (HTTP/1.1).

Le successive righe, nel formato header: valore, si chiamano intestazioni di richiesta e forniscono ulteriori informazioni come ad esempio i formati di risposta accettati dal client (Accept) o il sistema operativo e browser utilizzati (User-agent).

Ricevuta la richiesta, il server prepara la risposta e la invia indietro al browser. Un esempio di risposta HTTP per l'esempio precedente potrebbe essere questa

```
HTTP/1.1 200 OK
Date: Tue, 13 Aug 2019 21:23:53 GMT
Connection: keep-alive, Keep-Alive



Content-Encoding: br



Server: Apache



Content-Type: text/html; charset=UTF-8



<!DOCTYPE html>



<html lang="it">



    <head>



        <title>Guida PHP</title>



    </head>



    <body>



        <h1>Guida PHP</h1>



    </body>



</html>
```

La prima riga di una risposta HTTP contiene:

- la versione del protocollo (HTTP/1.1)
- il codice di stato (200)
- un messaggio relativo all'esito della risposta (OK).

Seguono le intestazioni di risposta nel formato header: valore, una riga vuota e il messaggio (body) vero e proprio, contenente l'HTML della risorsa richiesta. Il codice di stato 200 indica che la richiesta ha avuto esito positivo.

Ora che sappiamo come vengono scambiati i messaggi tra browser e sito web, vediamo come possiamo recuperare queste informazioni tramite PHP.

Variabili superglobali

PHP consente la gestione dei dati scambiati durante una richiesta HTTP attraverso le **variabili superglobali**. Si tratta di variabili di **tipo array** a cui è **possibile accedere da qualunque parte di un programma**, poiché hanno una visibilità (scope) globale.

Sono anche chiamate autoglobali in quanto disponibili automaticamente all'interno delle applicazioni.

Elenchiamo di seguito tali variabili:

1. $_SERVER: contiene tutto ciò che viene inviato dal server nella riposta HTTP.
2. $_GET: contiene i parametri passati tramite il metodo GET.
3. $_POST: contiene i parametri passati tramite il metodo POST.
4. $_REQUEST: contiene i valori degli array $_GET, $_POST e $_COOKIE.
5. $_COOKIE: contiene i **cookie** inviati dal client al server.
6. $_SESSION: contiene variabili registrate come **variabili di sessione**.
7. $_FILES: contiene informazioni sui file inviati tramite il metodo POST.

In questa guida analizziamo le prime 4 variabili superglobali: $_SERVER, $_GET, $_POST e $_REQUEST.

$_SERVER

La prima variabile che andiamo a vedere è $_SERVER, un **array associativo** contenente dati sulla richiesta HTTP più altre informazioni come il file PHP utilizzato, l'indirizzo IP, il metodo HTTP, etc...

Copiamo e incolliamo il seguente codice di esempio dentro un file rinominato test-superglobali.php e visualizziamo il contenuto tramite un browser, visitando l'indirizzo localhost/test-superglobali.php (è necessario aver installato **PHP in locale**).

```
<?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>”;
```

L'output del programma nel mio caso è il seguente

```
localhost

Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36

::1

HTTP/1.1

GET
```

Dove

- La riga 1 mostra il sito da dove è partita la richiesta. Essendo in locale viene mostrato localhost.
- La riga 2 mostra il tipo di dispositivo utilizzato (sistema operativo + browser).
- La riga 3 mostra l'indirizzo IP di chi ha effettuato la richiesta. In questo caso è mostrato l'indirizzo ::1 in versione IPV6 equivalente a 127.0.0.1 in versione IPV4.
- La riga 4 mostra la versione del protocollo utilizzato.
- La riga 5 mostra il metodo utilizzato, in questo caso GET.
- La riga 6 è vuota, in quanto non abbiamo passato nessun parametro nell'URL. Tra un pò vedremo come visualizzare questo valore.

La seguente tabella mostra l'elenco delle informazioni disponibili nell'array $_SERVER

Variabile	Valore
$_SERVER['DOCUMENT_ROOT']	La cartella radice dello script definita nel file di configurazione del server.
$_SERVER['HTTP_ACCEPT']	Il valore dell'intestazione Accept. Es. text/html
$_SERVER['HTTP_ACCEPT_CHARSET']	Il set di caratteri accettato. Es. utf-8
$_SERVER['HTTP_ACCEPT_ENCODING']	il tipo di encoding accettato. Es. gzip, deflate
$_SERVER['HTTP_ACCEPT_LANGUAGE']	La lingua accettata dal browser. Es. it-IT
$_SERVER[HTTP_CONNECTION']	Il valore dell'intestazione Connection. Es. keep-alive
$_SERVER['HTTP_HOST']	Il sito da dove è stata effettuata la richiesta.
$_SERVER['HTTP_REFERER']	L'indirizzo della pagina di provenienza, se esiste. Utile per sapere da dove proviene chi visita un sito.
$_SERVER['HTTP_USER_AGENT']	Informazioni sul sistema operativo e il browser web di chi ha effettuato la richiesta.
$_SERVER['HTTPS']	Impostato ad un valore non vuoto se la richiesta viene fatta tramite il protocollo HTTPS.
$_SERVER['QUERY_STRING']	La query string, se presente, contenente eventuali parametri passati alla pagina tramite GET.
$_SERVER['REMOTE_ADDR']	L'indirizzo IP del client che effettua la richiesta. Es. 127.0.0.1
$_SERVER['REMOTE_HOST']	L'indirizzo host del client che effettua la richiesta. Es. localhost

Variabile	Valore
$_SERVER['REMOTE_PORT]	La porta usata dal client per la connessione. Es. 80
$_SERVER['SCRIPT_FILENAME']	Il percorso assoluto dello script in esecuzione.
$_SERVER['SCRIPT_NAME']	Il percorso relativo dello script in esecuzione.
$_SERVER['SERVER_PORT']	La porta sulla quale il server ha accettato la richiesta.
$_SERVER['SERVER_SOFTWARE']	Informazioni sul server web e la versione di PHP in uso. Es. Apache/2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.5
$_SERVER['REQUEST_METHOD']	Il metodo HTTP della richiesta. Es. GET o POST.
$_SERVER['REQUEST_URI']	L'URI associato alla richiesta HTTP. Es. /index.php
$_SERVER['REQUEST_TIME']	La data/ora in cui è stata effettuata la richiesta. Es. Tue, 13 Aug 2019 20:35:00 GMT

$_GET

Nell'esempio precedente abbiamo utilizzato la variabile $_SERVER['QUERY_STRING'] per recuperare i parametri passati tramite il metodo GET e che verranno resi disponibili nell'array $_GET, la seconda variabile superglobale che andiamo ad analizzare.

Riprendiamo il nostro script test-superglobali.php e visualizziamone il contenuto digitando nella barra degli indirizzi del browser il seguente URL

localhost/test-superglobali.php?name=Mario&lastname=Rossi

Questa volta la riga con la variabile $_SERVER['QUERY_STRING'] non sarà vuota e conterrà il seguente valore

```
name=Mario&lastname=Rossi
```

Come è possibile recuperare singolarmente tali valori per poterli utilizzare nel programma? Risposta: attraverso l'array $_GET.

Creiamo un nuovo file e incolliamo il seguente codice

```
<?php

if (isset($_GET['name'])) {

    echo $_GET['name']."<br>";

}

if (isset($_GET['lastname'])) {
```

```
    echo $_GET['lastname']."<br>";


}
```

In questo esempio abbiamo utilizzato la funzione isset() del linguaggio. Questa funzione consente di verificare l'esistenza di una variabile (e che non sia uguale a NULL), infatti se proviamo a stampare direttamente i valori dell'array $_GET senza passare nessun parametro, otteniamo un avviso simile a questo

Notice: Undefined index: name in C:\xampp\htdocs\test-superglobali.php on line 2

Dunque la funzione isset() risulta fondamentale per scrivere codice più robusto.

$_POST

A differenza di GET, col metodo POST i dati sono inviati al server in maniera separata e non sono visibili nell'URL. In questo caso, PHP fornisce la variabile superglobale $ _POST per accedere a tutte le informazioni inviate tramite il metodo POST, oppure tramite un modulo HTML che utilizza POST come metodo.

Vediamo un esempio di codice (salviamo il seguente programma in un file con estensione .php e visualizziamo l'output nel browser).

```
<?php

if (isset($_POST['submit'])) {

    if (isset($_POST['name'])) {

        echo "Nome: ".$_POST['name']."<br>";

    }




 if (isset($_POST['lastname'])) {

        echo "Cognome: ".$_POST['lastname'];

    }

} else {
```

```
?>

<form action="phpinfo.php" method="post">

    <input type="text" name="name" value="Mario">

    <input type="text" name="lastname" value="Rossi">

    <input type="submit" name="submit" value="submit">

</form>

<?php

}

?>
```

Una volta cliccato sul pulsante "submit", lo script stampa a video, se esistono, i valori delle variabili $_POST['name'] e $_POST['lastname'].

Nell'esempio appena visto abbiamo usato il metodo POST, ma è possibile utilizzare GET al suo posto per recuperare le variabili dopo il submit. È sufficiente sostituire

```
method="post"
```

con

```
method="get"
```

e recuperare le variabili al submit con la superglobale $_GET.

Tuttavia GET ha delle limitazioni e risulta meno sicuro rispetto a POST, in quanto i parametri passati vengono accodati nell'URL e dunque sono visibili a tutti. Questo caso particolare non rappresenta un problema perché non stiamo passando dati sensibili (password), ma nel caso di un login ad un'area riservata va usato POST per effettuare l'autenticazione.

Nota: il metodo POST mette al sicuro i dati inviati al server solamente per una questione di visibilità, in quanto non visibili nell'URL. Tuttavia questi dati possono essere intercettati da un malintenzionato, in quanto col protocollo HTTP i dati sono trasferiti dal browser al server su

connessione non sicura. Per avere quindi una maggiore sicurezza è necessario utilizzare POST con la versione sicura di HTTP, ossia HTTPS, che utilizza un certificato SSL/TLS e la crittografia per proteggere i dati scambiati tra client e server. Un utente che compila un modulo di un sito web che utilizza HTTP (le URL iniziano per http://) verrà notificato dal browser tramite l'avviso "Non sicuro".

GET vs POST

Di seguito elenchiamo alcune importanti differenze tra i metodi GET e POST

	GET	POST
Clic sul pulsante back del browser	Non causa nessun effetto.	Il browser avvisa l'utente che i dati saranno reinviati se si aggiorna la pagina.
Preferiti browser	La risorsa può essere inserita tra i preferiti.	La risorsa non può essere inclusa tra i preferiti.
Copia cache	La risorsa può essere messa in cache.	La risorsa non può essere messa in cache.
Tipo di codifica	application/x-www-form-urlencoded	application/x-www-form-urlencoded oppure multipart/form-data per l'upload di file.
Restrizione sulla lunghezza dei dati inviati	2048 caratteri per un URL contenente i parametri.	Nessuna
Restrizione sui tipi di dati inviati	Sono ammessi solo caratteri ASCII.	Nessuna restrizione. Sono ammessi anche dati binari.
Sicurezza	I parametri vengono accodati nell'URL e sono visibili a chiunque.	È un po più sicuro di GET perché i parametri non sono visibili nell'URL e non vengono memorizzati nella cronologia del browser.

$_REQUEST

La variabile $ _REQUEST contiene i valori degli array $ _GET, $ _POST e $ _COOKIE. Parleremo della variabile $ _COOKIE quando analizzeremo i cookie.

La variabile $ _REQUEST può essere utilizzata per recupeare i dati inviati tramite GET e POST. Vediamo un esempio di utilizzo

```
<?php

if (isset($_POST['submit'])) {

    var_dump($_REQUEST);

} else {

?>

<form action="phpinfo.php?action=save" method="post">
```

```
    <input type="text" name="name" value="Mario">

    <input type="text" name="lastname" value="Rossi">

    <input type="submit" name="submit" value="submit">

</form>

<?php

}

?>
```

Dopo aver cliccato sul pulsante "submit" del modulo otteniamo il seguente output

```
array(4) {

  ["action"]=>

  string(4) "save"

  ["name"]=>

  string(5) "Mario"

  ["lastname"]=>

  string(5) "Rossi"

  ["submit"]=>

  string(6) "submit"

}
```

Come possiamo notare, l'array $_REQUEST contiene sia i valori salvati dell'array $_GET ('action'), sia quelli dell'array $_POST ('name', 'lastname' e 'submit').

Nelle prossime guide vedremo altre variabili molto utilizzate nelle applicazioni: $_COOKIE, $_SESSION e $_FILES.

Cookie: cosa sono e come gestirli in PHP

Per sua natura, il protocollo HTTP è stateless (senza stato), ossia non consente di conservare lo stato dell'utente tra una richiesta HTTP e la successiva.

Per ovviare a questo problema, gli sviluppatori web possono fare affidamento sul meccanismo dei cookie.

In questa guida vedremo **cosa sono i cookie e il loro funzionamento**, i casi di utilizzo e come crearli e accedervi in PHP attraverso la **variabile superglobale** $_COOKIE.

Indice dei contenuti

1. **Cosa sono i cookie**
2. **Funzionamento**
3. **Utilizzi**
4. **Creare un cookie**
5. **Accedere a un cookie**
6. **Modificare un cookie**
7. **Eliminare un cookie**

Cosa sono i cookie

I **cookie sono piccoli file di testo che consentono di memorizzare dati** sul computer dell'utente. Sono in genere utilizzati per tenere traccia dell'utente che visita per la prima volta un sito web e poter personalizzare le pagine quando lo stesso utente visiterà nuovamente quel sito.

Dato che i cookie possono contenere informazioni sensibili legate alla privacy utente, il loro utilizzo nei siti web è regolamentato da un provvedimento del Garante dell'8 maggio 2014 e dal **GDPR**, il regolamento generale sulla protezione dei dati entrato in vigore a maggio 2018.

Funzionamento

Tecnicamente i cookie sono degli **header HTTP** inviati dal server al client. Sono composti da una coppia chiave/valore e una serie di direttive opzionali.

Vediamo un esempio

```
Set-Cookie: session_id=1234;

Set-Cookie: session_id=1234; Expires=Fri, 30 Aug 2019 07:28:00 GMT;

Set-Cookie: session_id=1234; Max-Age=3600

Set-Cookie: session_id=1234; Domain=guidaphp.it

Set-Cookie: session_id=1234; Path=/
```

```
Set-Cookie: session_id=1234; Secure



Set-Cookie: session_id=1234; HttpOnly
```

L'intestazione Set-Cookie imposta un cookie di nome session_id sul browser dell'utente, assegnandogli il valore 1234.

Nelle righe successive viene impostato lo stesso cookie con alcune direttive opzionali che elenchiamo di seguito:

- Expires: imposta la data di validità del cookie. Se non specificato, il cookie sarà valido fino alla fine della sessione, ossia alla chiusura del browser.
- Max-age: è un altro modo di indicare il periodo di vita del cookie. Il valore in questo caso deve essere in secondi.
- Domain: specifica per quali domini verranno utilizzati i cookie.
- Path: indica il percorso URL associato al cookie.
- Secure: indica che i cookie verranno inviati solamente su connessione sicura (HTTPS).
- HttpOnly: indica che i cookie non sono accessibili via Javascript tramite document.cookie. Questo limita la possibilità di **attacchi di tipo XSS**.

L'esempio precedente può essere anche riscritto accorpando le direttive in un'unica riga

```
Set-Cookie: session_id=1234; Expires=Fri, 30 Aug 2019 07:28:00 GMT; Max-Age=3600; Domain=guidaphp.it; Path=/;
Secure; HttpOnly
```

Utilizzi

I cookie sono prevalentemente utilizzati per **funzionalità di autenticazione, tracciatura di sessioni e memorizzazione di informazioni** riguardanti gli utenti che accedono a un sito.

Nello specifico i cookie sono utilizzati per:

- Memorizzare il login utente ad un'area riservata. È la tipica spunta "ricordami" che vediamo ad esempio su alcuni forum.
- Personalizzare l'esperienza di navigazione sulla base delle preferenze dell'utente. Ad esempio, su siti multilingua è possibile memorizzare la lingua dell'utente (ricavata tramite la variabile globale $_SERVER['HTTP_ACCEPT_LANGUAGE']) e, nelle successive visite, reindirizzare l'utente alla versione nella sua lingua.
- Tenere traccia dei prodotti del carrello della spesa su siti eCommerce.
- Tracciare i siti già visitati da un utente e ricavare informazioni da usare per mostrargli banner pubblicitari in linea con le sue preferenze.
- Tracciare il numero di visite dello stesso utente ad un sito web.

Nota: è assolutamente sconsigliato archiviare dati sensibili nei cookie poiché potrebbero essere manipolati da utenti malintenzionati. Per memorizzare in modo sicuro dati sensibili durante la navigazione utente è necessario utilizzare le sessioni.

Creare un cookie

In PHP possiamo **creare un cookie** tramite la funzione setcookie(), la cui sintassi è la seguente:

```
setcookie(string $name, string $value = "", int $expire = 0, string $path = "", string $domain = "", bool $secure =
false, bool $httpOnly = false);
```

e dove:

- $name è il nome del cookie. Deve essere di tipo stringa ed è l'unico parametro obbligatorio, i successivi sono opzionali.
- $value è il valore del cookie. Deve essere di tipo stringa.
- $expire è la data di scadenza nel formato **Unix timestamp**. Trascorso tale tempo i cookie saranno inaccessibili. Il valore di default è 0.
- $path specifica il percorso in cui il cookie sarà disponibile. Se impostato su /, il cookie sarà disponibile all'interno dell'intero dominio.
- $domain specifica il dominio in cui il cookie è disponibile, ad esempio www.dominio.it
- $secure se impostato a 1, specifica che il cookie può essere inviato solo tramite HTTPS.
- $httpOnly è la modalità che rende il cookie non accessibile via Javascript.

Facciamo un esempio reale

```
<?php

$expire = 30*24*60*60;

setcookie("lang", "it", time() + $expire);
```

Nell'esempio precedente abbiamo creato un cookie di nome lang con la funzione setcookie() e gli abbiamo assegnato il valore it.

Abbiamo quindi impostato la scadenza tramite la funzione nativa di PHP time(), che ritorna il tempo attuale in timestamp, a cui abbiamo aggiunto un numero di secondi equivalente a 1 mese di 30 giorni (30 giorni per 24 ore per 60 minuti per 60 secondi).

> **Nota:** Essendo un header, il cookie deve essere impostato prima di qualsiasi output HTML (anche una riga vuota), altrimenti viene generato un avviso come il seguente **

Warning: Cannot modify header information - headers already sent by...

** solo se la direttiva output_buffering del php.ini è impostata a Off.

Accedere a un cookie

Una volta creato un cookie, questo sarà accessibile tramite la **variabile superglobale $_COOKIE**, che è di **tipo array** come $_GET e $_POST.

Prendendo spunto dall'esempio precedente possiamo scrivere così

```
<?php




if ($_COOKIE["lang"] == 'it') {

    header('Location: /it');

} elseif ($_COOKIE["lang"] == 'en') {

    header('Location: /en');

} else {

    header('Location: /');

}

exit; // termina l'esecuzione dello script
```

La funzione header() di PHP consente l'invio di intestazioni HTTP al browser. In questo caso abbiamo utilizzato l'intestazione Location, che consente di reindirizzare l'utente verso l'URL specificato dopo i due punti.

Nel caso di un utente che accede per la prima volta a un sito multilingua, sapremo a quale versione del sito reindirizzarlo alla visita successiva, tramite il cookie memorizzato nella variabile $_COOKIE["lang"].

> **Nota:** è consigliabile utilizzare la funzione isset() prima di accedere al valore di un cookie. La funzione isset() verifica l'esistenza di una variabile e che non sia impostata a NULL.

Modificare un cookie

Per modificare un cookie possiamo semplicemente sovrascrivere il vecchio valore con il nuovo tramite la funzione setcookie()

```
<?php

setcookie("lang", "it");

setcookie("lang", "en"); // Qui stiamo sovrascrivendo il valore del cookie di nome lang
```

Eliminare un cookie

Per **eliminare un cookie** è sufficiente impostare il suo valore a "" (stringa vuota), oppure a 0 o qualsiasi valore che PHP interpreta come false.

```
<?php

secookie('lang', '');
```

Per essere certi di eliminare il cookie, dobbiamo passare il parametro $expire con un valore che fa riferimento al passato, come mostrato di seguito

```
<?php

$expire = time() -3600; // impostiamo $expire a un'ora fa

secookie('lang', '', $time);
```

Ora che sappiamo cos'è un cookie e come utilizzarlo nelle nostre applicazioni web, nella prossima guida analizzeremo un'altra variabile globale di PHP: la variabile $_SESSION.

Sessioni PHP: cosa sono e come funzionano, casi d'uso ed esempi

Sebbene PHP consenta di memorizzare lo stato utente tra una pagina web e l'altra attraverso i **cookie**, questo meccanismo presenta problemi di sicurezza.

Infatti, poiché i cookie sono memorizzati sul computer dell'utente, è possibile che un utente malintenzionato modifichi facilmente questi dati al fine di compromettere l'intera applicazione.

Un modo alternativo per rendere accessibili i dati in maniera permanente attraverso le pagine di un sito web, consiste nell'utilizzare una **sessione PHP**.

Prima di dare uno sguardo a come implementare le sessioni in PHP, vediamo innanzitutto cosa sono, come funzionano e in quali scenari utilizzare questo meccanismo di memorizzazione.

Indice dei contenuti

1. **Cosa sono le sessioni e come funzionano in PHP**
2. **Casi d'uso**
3. **Inizializzare una sessione**
4. **Accedere ai dati di una sessione**
5. **Modificare i dati di una sessione**
6. **Terminare una sessione**
7. **Differenze tra sessioni e cookie**

Cosa sono le sessioni e come funzionano in PHP

Le sessioni sono un insieme di **informazioni memorizzate in un file** che si trova sul server dove gira la nostra applicazione.

Quando viene avviata una sessione, PHP crea un **identificatore univoco** per quella particolare sessione, denominato PHPSESSID e rappresentato da una stringa casuale generata tramite una **funzione di hash**.

Allo stesso modo, viene creato e memorizzato sul server un file temporaneo all'interno di una directory designata al salvataggio delle sessioni.

Quando uno script PHP desidera **recuperare il valore da una variabile di sessione**, PHP preleva il valore dell'identificatore di sessione PHPSESSID e lo confronta con quello memorizzato sul server per vedere se sono uguali. In caso affermativo la sessione continua, altrimenti viene interrotta.

Una sessione può anche terminare quando l'utente chiude il browser o dopo aver fatto il logout dalla propria area riservata, oppure si interrompe automaticamente dopo un certo periodo di inattività.

> **Nota:** La posizione della directory contenente i file di sessione è determinata da un'impostazione del file **php.ini** chiamata session.save_path.

Casi d'uso

Le sessioni PHP vengono utilizzate per implementare soprattutto le seguenti funzionalità:

- **Area riservata sito**: nella gestione di un **sistema di autenticazione tramite login e password**, si è soliti creare una variabile di sessione da utilizzare in tutte le pagine che devono mostrare un contenuto protetto.
- **Carrello spesa sito eCommerce**: tramite sessioni è possibile tenere traccia dei prodotti inseriti nel carrello tra una pagina e l'altra, continuando la navigazione del sito fino al checkout.
- **Messaggi flash**: dopo un'operazione di modifica dei dati di un'applicazione (creazione, modifica, etc...) si è soliti redirezionare l'utente ad una pagina che mostri un messaggio temporaneo di errore o successo. Con le sessioni è possibile farlo in maniera semplice.

Vediamo adesso come creare una sessione e come accedere ai dati di quella sessione, modificarli oppure cancellarli del tutto.

Inizializzare una sessione

Per memorizzare un dato in una sessione in PHP è necessario **inizializzare la sessione** richiamando la funzione session_start() all'inizio dello script.

> **Nota:** la funzione session_start() deve essere presente in ciascuno script PHP in cui vogliamo gestire le sessioni.

Una volta che la sessione è inizializzata, è possibile utilizzare la **variabile superglobale** $_SESSION per memorizzare variabili di sessione. Questa non è altro che un **array associativo**, quindi è sufficiente specificare un indice, che sarà il nome della variabile di sessione, e un valore.

Vediamo un esempio di utilizzo

```
session_start();

$_SESSION['username'] = 'rs200';
```

In questo caso abbiamo assegnato un valore di tipo stringa alla variabile di sessione $_SESSION['username'].

Accedere ai dati di una sessione

Una volta creata una variabile di sessione, recuperarne il contenuto è molto semplice in quanto è possibile accedervi come se fosse un normale array associativo.

Modificando l'esempio precedente, possiamo scrivere

```
<?php

session_start();
```

```
if (!isset($_SESSION['username'])) {

    header('Location:login.php');

    exit;

} else {

    echo "Benvenuto utente ".$_SESSION['username'];

}

// Codice area riservata
```

Nello script appena visto eseguiamo un controllo sulla variabile $_SESSION['username'] per vedere se è stata inizializzata (funzione isset()).

In caso negativo l'utente viene redirezionato alla pagina di login, in caso positivo la navigazione nell'area riservata prosegue con la stampa di un messaggio.

Modificare i dati di una sessione

Per **modificare una variabile di sessione** è sufficiente sovrascrivere il suo contenuto, riassegnando un nuovo valore alla variabile

```
<?php

session_start();

$_SESSION['username'] = 'rs200';

$_SESSION['username'] = 'f40';

echo $_SESSION['username']; // Stampa f40
```

Terminare una sessione

Per **terminare una sessione** possiamo usare la funzione session_destroy() che distrugge completamente una sessione

```
<?php

session_start();

if (isset($_POST['logout'])) {

    session_destroy();

    header('Location:login.php');

    exit;

}
```

Nota: Prima di distruggere una sessione è necessario richiamare la funzione session_start().

Se è necessario **eliminare il contenuto di una singola variabile di sessione** senza far terminare la sessione stessa, è possibile utilizzare la funzione unset() in questo modo

```
<?php

session_start();

$_SESSION['username'] = 'rs200';

unset($_SESSION['username']);

if (!isset($_SESSION['username'])) {

    echo "La variabile di sessione non esiste più";

}
```

Differenze tra sessioni e cookie

Per concludere questa guida sulle sessioni PHP, mettiamole a confronto con i cookie tramite una tabella

Sessioni	Cookie
Le sessioni sono file memorizzati sul server che contengono informazioni sull'utente	I cookie sono file memorizzati sul client che contengono informazioni sull'utente
La durata massima di una sessione è configurabile in secondi tramite la direttivasession.gc_maxlifetime del php.ini	La durata massima del cookie viene impostata tramite la funzione setcookie(), aggiungendo un valore positivo al parametro $expire. Es. setcookie("lang", "it", time() + 3600)
Le variabili di sessione vengono gestite tramite la superglobale $_SESSION	I cookie vengono gestiti tramite la superglobale $_COOKIE
Prima di poter accedere alle variabili di sessione è necessario utilizzare session_start()	Poiché risiedono sul computer utente, è possibile accedere ai cookie senza dover inizializzare nulla
Non ci sono limiti per quanto concerne la dimensione massima di una sessione. Dipende dallo spazio sul server.	Ciascun cookie può avere una dimensione massima di 4KB
Per cancellare una variabile si usa unset(). Per distruggere una sessione si una session_destroy()	Per cancellare un cookie è necessario impostare la durata con un valore che fa riferimento al passato. Es: setcookie("lang", "it", time() - 3600)
Una sessione termina alla chiusura del browser da parte dell'utente oppure automaticamente dopo che è passato un periodo di inattività in secondi pari al valore impostato in session.gc_maxlifetime	Un cookie termina quando viene eliminato dall'utente tramite le impostazioni del browser oppure quando termina la durata impostata

Upload di file in PHP e variabile $_FILES

Come altri linguaggi di programmazione, anche PHP consente il caricamento di **file** su server tramite un **form HTML**. In questa guida vedremo come effettuare l'**upload di file in PHP** e come recuperare le informazioni relative a tali file per utilizzarle nelle nostre applicazioni.

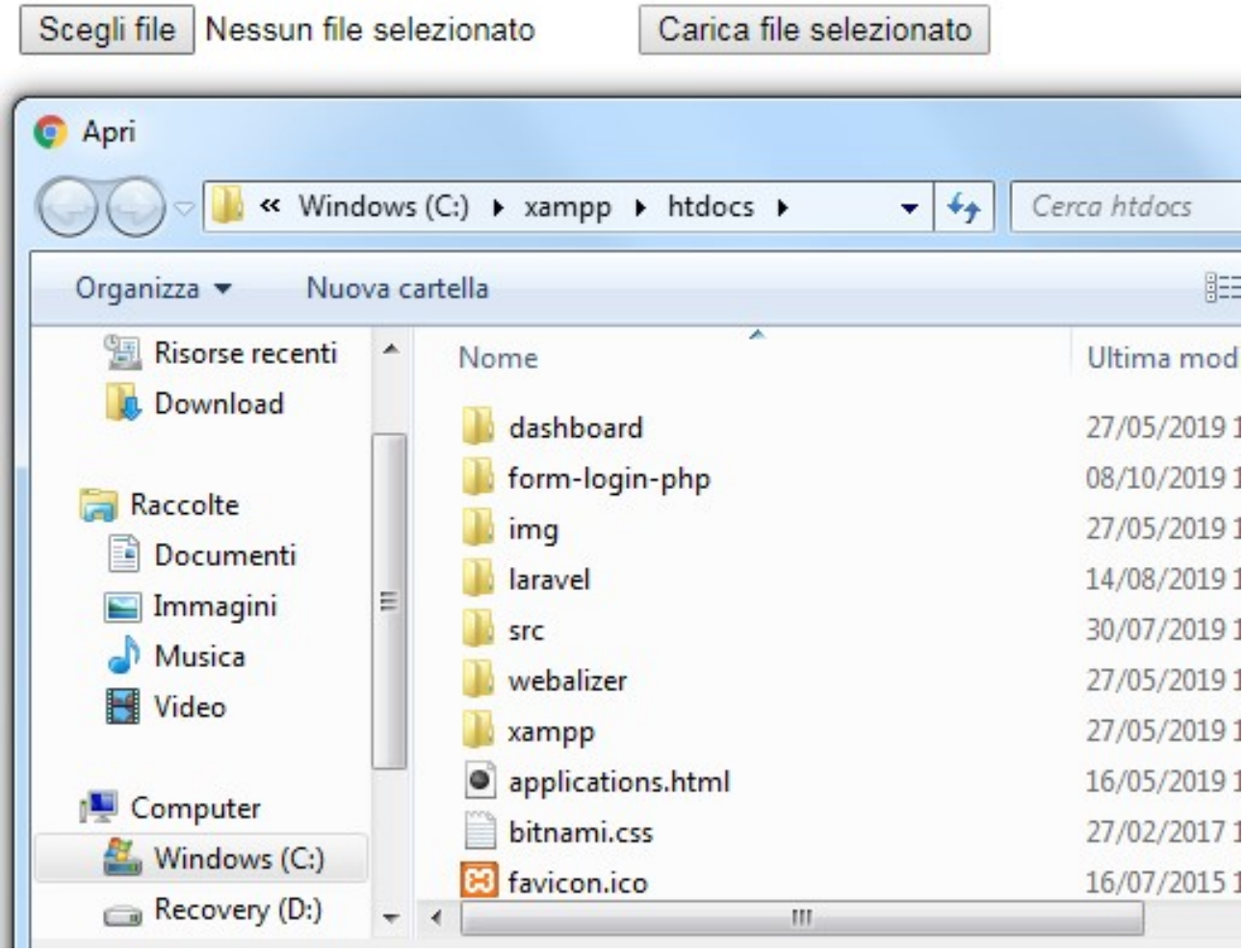
Indice dei contenuti

1. Come funziona il caricamento
2. Il formato multipart/form-data
3. Variabile $_FILES
4. Funzione move_uploaded_file
5. Upload multiplo
6. Errori comuni
7. Impostazioni php.ini per upload file

Come funziona il caricamento

Prima di analizzare un po' di codice, vediamo quali sono i passaggi che avvengono durante il caricamento di un file:

1. L'utente accede alla pagina contenente un modulo HTML con un input per selezionare file e un pulsante "Carica file".
2. L'utente fa clic sul pulsante "Scegli file" e seleziona un file dal suo computer.
3. Il percorso del file selezionato appare a lato del pulsante "Scegli file", quindi l'utente fa clic sul pulsante di caricamento.
4. Il file selezionato viene inviato al server e memorizzato in una directory temporanea.
5. Lo script PHP designato per la gestione del file caricato verifica che il file sia stato caricato con successo, quindi lo copia in una directory specifica.



Vediamo nello specifico cosa occorre per riprodurre i passaggi elencati sopra.

Il formato multipart/form-data

Il **protocollo HTTP** consente il **caricamento di file sul server** tramite l'utilizzo del formato *multipart/form-data* e dell'input di tipo *file*.

Il **formato multipart/form-data** garantisce che i dati (i file) inviati siano codificati nel formato **MIME** *multipart*, utilizzato anche nell'invio delle email e necessario per trasferire dati binari come immagini, video, file zippati, etc...

L'esempio seguente mostra la struttura del form da utilizzare per l'upload di file

```
<form action="upload.php" method="post" enctype="multipart/form-data">

    <input type="file" name="file">

    <input type="submit" name="upload" value="Carica file">

</form>
```

Come possiamo vedere, il valore dell'attributo method è post, che è il metodo HTTP utilizzato per l'**invio di file al server**. L'action contiene il valore upload.php, ossia lo script che si occuperà di gestire il file caricato (lo vedremo nel paragrafo successivo).

Il tag input è di tipo *file* e consente di mostrare nel browser utente una casella di testo con un pulsante "Seleziona file" (o altra etichetta a seconda del browser utilizzato). È possibile caricare qualsiasi tipo di file, ad esempio immagini, video, file compressi, documenti di Word o PDF e tanti altri tipi di file.

> **Nota:** come vedremo in un tutorial dedicato alla creazione di un form di upload sicuro, è consigliabile verificare il tipo di file caricato per evitare che un utente malintenzionato possa trasferire sul server file dannosi (ad esempio eseguibili).

Una volta selezionato un file dal sistema operativo, per completare l'operazione di upload l'utente deve cliccare sul pulsante "Carica file".

Vediamo adesso come recuperare le informazioni del file in PHP.

Variabile $_FILES

PHP è in grado di **recuperare un file inviato al server** tramite la **variabile superglobale** $_FILES. Essendo un **array associativo**, possiamo stampare il suo contenuto tramite print_r().

Nel file upload.php andremo ad inserire il seguente codice PHP

```
<?php

print_r($_FILES);
```

che produce un output simile a questo

```
Array

(

    [file] => Array

        (

            [name] => immagine.jpg

            [type] => image/jpeg

            [tmp_name] => C:\xampp\tmp\php599B.tmp

            [error] => 0

            [size] => 23459

        )
```

L'**array** $_FILES contiene un solo file identificato dall'indice *file* (l'attributo name nel modulo HTML).

Le informazioni relative a tale file sono le seguenti:

1. name: il nome del file, nell'esempio immagine.jpg.
2. type: il tipo di file nel formato MIME. Nell'esempio è image/jpeg in quanto abbiamo caricato un file di tipo immagine.
3. tmp_name: il percorso della cartella temporanea. Ad ogni upload, PHP genera un nome casuale e lo assegna al file caricato in questa cartella. Es. php599B.tmp.
4. error: l'errore relativo al caricamento, nel nostro caso 0 equivale a un successo.
5. size: la dimensione in byte del file caricato.

Funzione move_uploaded_file

Possiamo utilizzare le informazioni presenti nell'array $_FILES per **memorizzare il file caricato in una directory** specifica. È possibile farlo grazie alla funzione move_uploaded_file().

Vediamo un esempio pratico

```
<?php

$uploadDir = __DIR__.'\uploads';

foreach ($_FILES as $file) {

    if (UPLOAD_ERR_OK === $file['error']) {

        $fileName = basename($file['name']);

        move_uploaded_file($file['tmp_name'], $uploadDir.DIRECTORY_SEPARATOR.$fileName);

    }

}
```

Spiegazione codice

1. Alla riga 1 definiamo la directory definitiva nella quale salveremo i file caricati, quindi cicliamo l'array $_FILES con un **foreach**.
2. Alla riga 5 verifichiamo che i file siano stati inviati con successo. In questo caso utilizziamo l'**operatore di confronto** === per fare un confronto stretto. Se la variabile $file['error'] contiene il valore 0 (che equivale alla costante UPLOAD_ERR_OK) significa che l'invio è andato a buon fine.
3. Se la condizione alla riga 5 viene soddisfatta, il file temporaneo viene spostato all'interno della directory $uploadDir, tramite la funzione move_uploaded_file().

Nota: abbiamo utilizzato la costante predefinita DIRECTORY_SEPARATOR come separatore delle cartelle. A seconda del sistema operativo in uso, la costante sarà uguale a un backslash "\" (Windows) o a uno slash "/" (Linux).

Alla riga 6 abbiamo utilizzato la funzione basename() per ricavare il nome del file. Questa funzione elimina eventuali percorsi relativi o assoluti presenti nel nome del file. Questo evita il cosiddetto **directory traversal**, una tipologia di attacco in cui un utente malintenzionato sfrutta una falla dell'applicazione per navigare il file system del server.

> **Nota:** la directory dove verrano salvati i file deve avere i permessi di scrittura abilitati. Su sistemi Linux i permessi devono essere settati a 0755.

Upload multiplo

Lo script precedente consente di caricare un file alla volta. Tramite una semplice modifica al codice HTML del modulo, è possibile **caricare più file** con una sola richiesta.

Abbiamo a disposizione due modi per fare questo:

- Aggiungere tanti tag input quanti sono i file che vogliamo caricare.
- Aggiungere al tag l'attributo multiple e le parentesi quadre [].

Nel primo caso otteniamo il seguente codice HTML

```
<form action="upload.php" method="post" enctype="multipart/form-data">

    <input type="file" name="file1">

    <input type="file" name="file2">

    <input type="file" name="file3">

    <input type="submit" name="upload" value="Carica file">

</form>
```

mentre nel secondo caso questo codice

```
<form action="upload.php" method="post" enctype="multipart/form-data">

    <input type="file" name="file[]" multiple> <!-- riga modificata per upload multipli -->

    <input type="submit" name="upload" value="Carica file">

</form>
```

L'attributo multiple consente la selezione multipla di file (tenere premuto il tasto Ctrl mentre si clicca sui file). Alla fine del caricamento, saranno disponibili nel solito array $_FILES.

Prima di ciclare l'array $_FILES per poter salvare i file nella directory, dobbiamo modificare la sua struttura.

Possiamo editare il codice del file upload.php in questo modo

```
<?php

$uploadDir = __DIR__.'\uploads';

$files = rearrange($_FILES);

foreach ($files as $file) {

    if (UPLOAD_ERR_OK === $file['error']) {

        $fileName = basename($file['name']);

        move_uploaded_file($file['tmp_name'], $uploadDir.DIRECTORY_SEPARATOR.$fileName);

    }

}

function rearrange($files)

{

    foreach($files as $key1 => $val1) {

        foreach($val1 as $key2 => $val2) {

            for ($i = 0, $count = count($val2); $i < $count; $i++) {

                $newFiles[$i][$key2] = $val2[$i];
```

```
            }

        }

    }

    return $newFiles;

}
```

Abbiamo aggiunto la **funzione** personalizzata rearrange(), che consente di riorganizzare gli elementi dell'array $_FILES trasformandolo in un array bidimensionale più comodo da ciclare.

Per approfondire segui questo **link**.

Errori comuni

Durante il caricamento dei file, è possibile imbattersi in qualche errore in fase di esecuzione dello script.

Un tipico errore è il seguente

Warning: move_uploaded_file(...): failed to open stream: No such file or directory in...

Soluzione: questo errore si verifica se il percorso del file nella cartella di destinazione non è valido. Potrebbe essere necessario creare la cartella di destinazione dei file oppure correggere il percorso di destinazione.

Impostazioni php.ini per upload file

Concludiamo questa guida analizzando alcune delle direttive del file **php.ini** che consentono di **cambiare i settaggi relativi all'upload di file**

1. file_uploads: determina se consentire o meno il caricamento di file.
2. upload_max_filesize: specifica la dimensione massima che può avere un singolo file prima di essere caricato.
3. post_max_size: imposta la dimensione massima consentita per i dati inviati tramite POST. Deve essere maggiore di upload_max_filesize.
4. upload_tmp_dir: la directory dove vengono salvati temporaneamente i file caricati, in attesa di spostarli nella cartella definitiva.
5. max_file_uploads: il numero massimo di file che possono essere caricati tramite una singola richiesta HTTP.

Con la variabile $_FILES chiudiamo la serie di guide sulle superglobali di PHP. Il prossimo argomento sarà incentrato sulla gestione del file system e le operazioni che è possibile fare su file e directory.

Gestione dei file in PHP: le funzioni base

PHP mette a disposizione due modi per **memorizzare informazioni in maniera persistente** su disco: file e database. In questa guida analizziamo i **file**, mentre rimandiamo l'argomento **database** in una prossima guida dedicata.

I file possono contenere qualsiasi **tipo di dati**, oltre che numerose informazioni sui file stessi. Il linguaggio offre numerose funzioni per la **gestione del file system**.

Questa guida è dedicata in particolare all'apertura, lettura, scrittura e chiusura di file su disco. Nella guida successiva vedremo invece le **funzioni** del linguaggio per gestire le directory ed ulteriori funzioni per la gestione dei file.

Prima di andare avanti, introduciamo un concetto fondamentale per poter capire come lavorare con i file in PHP: il puntatore al file.

Indice dei contenuti

1. **Puntatore al file**
2. **Aprire un file con fopen()**
3. **Chiudere un file con fclose()**
4. **Modalità di apertura file**
5. **Funzione file_exists()**
6. **Leggere un file con fread()**
7. **Scrivere in un file con fwrite()**
8. **Funzioni file_get_contents() e file_put_contents()**

Puntatore al file

Il **puntatore al file** o descrittore di file è un numero intero assegnato al file al momento della sua apertura. Esso consente di riferirsi univocamente a tale file fino alla sua chiusura.

Dopo l'apertura, per poter effettuare operazioni sul file non dobbiamo più usare il suo nome ma fare riferimento al suo puntatore.

Per operare sui file è necessario avere accesso al loro *descrittore*, un record logico contentente informazioni sul file stesso:

- nome del file
- tipo del file
- la sua dimensione
- informazioni sui permessi di accesso al file
- la data di creazione
- la data dell'ultima modifica

Tali descrittori sono memorizzati su disco fisso in apposite aree chiamate *directory* (o tabelle dei descrittori), che sono file a loro volta.

Le **directory** hanno una struttura gerarchica rappresentabile come un albero, dove la directory principale è la radice (root) e via via scendendo ci sono le altre directory.

Per poter compiere operazioni sui file è necessario quindi caricare in memoria il descrittore del file, operazione che prende il nome di apertura del file.

Aprire un file con fopen()

Per **aprire un file in PHP** possiamo utilizzare la funzione fopen(), come mostrato nel seguente esempio

```
<?php

$filename = 'file.txt';

$handler = fopen($filename, 'r+');

if (false === $handler) {

    printf('Impossibile aprire il file %s', $filename);

    exit;

}

fclose($handler);
```

La funzione fopen() prende due parametri:

1. il file che vogliamo aprire
2. la modalità di apertura

e restituisce il **puntatore al file aperto** (nell'esempio abbiamo memorizzato tale puntatore nella variabile $handler). La funzione restituisce false in caso di errore nell'apertura del file.

Chiudere un file con fclose()

Dopo aver terminato di lavorare su un file, questo deve essere chiuso in qualche modo poiché occupa risorse e appesantisce il carico del server.

In PHP è la funzione fclose() che si occupa di **chiudere un file** per liberare la memoria. Essa prende come parametro il puntatore al file, come mostrato alla riga 10 dell'esempio precedente e ritorna true in caso di successo.

> **Nota:** sebbene PHP chiuda automaticamente tutti i file aperti al termine dell'esecuzione degli script, è buona prassi per un programmatore chiuderli all'interno dell'applicazione stessa per renderli disponibili ad altri processi o script che ne facciano richiesta.

Modalità di apertura file

Nell'esempio precedente abbiamo aperto il file in modalità r+, ossia in lettura e scrittura.

Nella seguente tabella sono riportate le possibili **modalità di apertura di un file** con fopen()

Modalità	Descrizione
r	Apre il file in lettura posizionando il puntatore all'inizio del file
r+	Apre il file in lettura e scrittura posizionando il puntatore all'inizio del file
w	Apre il file in scrittura posizionando il puntatore all'inizio del file. Cancella il contenuto del file. Se il file non esiste lo crea.
w+	Apre il file in lettura e scrittura posizionando il puntatore all'inizio del file. Cancella il contenuto del file. Se il file non esiste lo crea.
a	Apre il file in scrittura posizionando il puntatore alla fine del file. Se il file non esiste lo crea.
a+	Apre il file in lettura e scrittura posizionando il puntatore alla fine del file. Se il file non esiste lo crea.
x	Crea e apre un file in scrittura posizionando il puntatore all'inizio del file. Ritorna false e genera un E_WARNING se il file esiste già.
x+	Crea e apre un file in lettura e scrittura posizionando il puntatore all'inizio del file. Ritorna false e genera un E_WARNING se il file esiste già.
c	Apre il file in scrittura e, se non esiste, lo crea. Se il file esiste già il contenuto esistente viene preservato e il puntatore posizionato all'inizio del file. Questa modalità è utile nel caso di lock su file. Per approdondire leggi la documentazione per **flock()**
c+	Apre un file in lettura e scrittura. Vedi modalità c.

Funzione file_exists()

Se proviamo ad aprire un file che non esiste, PHP genera un messaggio di avviso simile al seguente

PHP Warning: fopen(file.txt): failed to open stream: No such file or directory in...

Per evitare ciò, PHP mette a disposizione la funzione file_exists() che **verifica l'esistenza di un file** ritornando true oppure false.

Di seguito un esempio

```
<?php

$filename = 'files.txt';

if (false === file_exists($filename)) {

    printf('Il file %s non esiste', $filename);
```

```
}
```

Leggere un file con fread()

La funzione fread() consente di **leggere dal file un numero definito di byte** e li restituisce sotto forma di **stringa** di testo.

La funzione accetta due parametri: il puntatore al file e il numero di byte da leggere.

Vediamo un esempio pratico

```
<?php

$filename = 'file.txt';

$handler = fopen($filename, 'r');

if (false === $handler) {

    printf('Impossibile aprire il file %s', $filename);

    exit;

}

$size = 1024;

while (!feof($handler)) {

    $content = fread($handler, $size);

    echo $content;

}

fclose($handler);
```

Dopo aver verificato l'esistenza del file, tramite un **ciclo while** leggiamo l'intero contenuto del file verificando ad ogni ciclo che non sia stata la raggiunta la **fine del file** tramite la funzione feof().

In questo esempio abbiamo utilizzato il valore 1024 come secondo parametro per la funzione fread(). Un valore più alto consentirebbe un'esecuzione più rapida dello script, ma consumerebbe più RAM ad ogni ciclo.

È dunque importante verificare di avere sufficiente memoria a disposizione prima di incrementare il numero di byte da estrarre dal file.

Scrivere in un file con fwrite()

Per **scrivere dati in un file** possiamo utilizzare la funzione fwrite().

La funzione accetta due parametri:

1. il puntatore al file
2. la stringa di testo da scrivere nel file

Se l'operazione avviene correttamente la funzione ritorna il numero di byte scritti, altrimenti false.

Di seguito è mostrato un esempio

```
<?php

$filename = 'file.txt';

$handler = fopen($filename, 'w+');

if (false === $handler) {

    printf('Impossibile aprire il file %s', $filename);

    exit;

}

for ($i = 1; $i <= 100; $i++) {

    fwrite($handler, "Riga $i\n"); // Scrive la stringa nel file 100 volte

}
```

```
fclose($handler);
```

Lo script scrive nel file 100 volte la stringa "Riga $i", con la variabile $i che viene incrementata di una unità ad ogni iterazione.

Avendo utilizzato la modalità di apertura w+, tutto il contenuto precedente del file verrà perso. Se il file non esiste, verrà creato.

Funzioni file_get_contents e file_put_contents

Per leggere e scrivere informazioni in un file, il linguaggio offre due alternative alle funzioni appena viste: file_get_contents() per **leggere il contenuto di un file** e file_put_contents() per **scrivere in un file**.

La funzione file_get_contents() prende come parametro il nome del file e restituisce il numero di byte in esso contenuti in formato stringa. In caso di errore la funzione ritorna false.

Ecco un esempio di utilizzo

```
<?php

$filename = 'file.txt';

$content = file_get_contents($filename);

if (false === $content) {

    printf('Impossibile leggere il file %s', $filename);

    exit;

}

print_r($filename);
```

La funzione file_put_contents() prende in input due parametri:

1. il percorso del file in cui scrivere
2. la stringa da inserire nel file

```php
<?php

$filename = 'file.txt';

$text = 'Questo è un test';

$bytes = file_put_contents($filename, $text);

if (false === $bytes) {

    printf('Impossibile scrivere nel file %s', $filename);

    exit;

}

print_r($bytes);
```

La funzione file_get_contents() può essere utilizzata per leggere il codice sorgente di una pagina web, come mostrato nell'esempio seguente

```php
<?php

$homepage = file_get_contents('https://www.google.it/');

echo $homepage;

// <!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="it">...
```

Nella guida successiva faremo una panoramica delle funzioni PHP più usate per eseguire operazioni su file.

Gestione dei file in PHP: funzioni avanzate

Dopo aver analizzato le funzioni che consentono le **operazioni di base sui file**, in questa guida vediamo quelle per effettuare operazioni più complesse come ad esempio **copiare, rinominare ed eliminare un file**, ottenere informazioni su un file e le funzioni per la gestione di cartelle.

Come già sappiamo, per poter lavorare con un file abbiamo bisogno di:

1. Aprire il file con fopen(), che ci restituisce il puntatore al file.
2. Effettuare le operazioni sul file utilizzando il suo puntatore.
3. Chiudere il file con fclose() utilizzando il suo puntatore.

Indice dei contenuti

1. Copiare un file
2. Rinominare o spostare un file
3. Eliminare un file
4. Verificare l'esistenza di un file
5. Ottenere informazioni
6. Leggere uno o più caratteri
7. Accesso casuale ai dati
8. Funzioni per gestire directory
9. Creare una cartella
10. Cancellare una cartella
11. Leggere il contenuto di una cartella
12. Rinominare una cartella

Copiare un file

PHP mette a disposizione la funzione copy() per **effettuare la copia di un file**. Può essere utilizzata per fare piccole copie di backup ad intervalli di tempo prestabiliti.

La funzione accetta due argomenti di **tipo stringa** che fanno riferimento rispettivamente al percorso del file di origine e a quello di destinazione.

Nel seguente script di esempio, la funzione copia il file file.txt che si trova nella directory corrente nella posizione indicata in $newFile, restituendo false in caso di errore.

```
<?php

$file = 'file.txt';

$newFile = 'file-copied.txt';
```

```
if (false === copy($file, $newFile)) {

    printf("Impossibile copiare il file %s", $file);

}
```

Nota: Se avessi voluto copiare il file di origine verso una directory diversa da quella corrente, avremmo dovuto specificare un valore diverso in $newFile come ad esempio "directory/file-copied.txt".

Rinominare o spostare un file

L'operazione di **spostamento di un file**, ossia quella che di solito eseguiamo tramite le combinazioni di tasti Ctrl + X (Copia) e Ctrl + V (Incolla), è possibile effettuarla tramite la funzione rename(), che come suggerisce il nome consente di **rinominare un file**.

La funzione prende in input il nome del file da rinominare ritornando true in caso di successo, false altrimenti.

```
<?php

$oldFile = '/old-path/image.jpg';

$newFile = '/new-path/new-image.jpg';

if (false === rename($oldFile, $newFile)) {

    printf('Impossibile rinominare il file %s', $oldPath);

}
```

Eliminare un file

La funzione unlink() consente di **eliminare un file** in maniera permanente dal file system. La funzione prende un singolo parametro, il nome del file che si vuole cancellare.

```
<?php

$file = 'file.txt';
```

```
if (false === unlink($file)) {

    printf("Impossibile eliminare il file %s", $file);

}
```

Verificare l'esistenza di un file

Le **funzioni PHP** viste fino ad ora potrebbero generare **messaggi di errore**, dovuti solitamente al fatto che il file sul quale vogliamo operare non esiste.

Non è insolito infatti imbattersi in un avviso simile al seguente

PHP Warning: fopen(file.txt): failed to open stream: No such file or directory in...

Per **verificare se un file esiste** o meno, il linguaggio mette a disposizione la funzione file_exists() che, preso il nome del file ne verifica l'esistenza restituendo true oppure false.

Riprendendo l'esempio relativo alla cancellazione di un file, per rendere il codice più robusto possiamo scrivere quanto segue

```
<?php

$file = 'file.txt';

if (false === file_exists($file)) {

    printf("Il file %s non esiste", $file);

    exit;

}

if (false === unlink($file)) {

    printf("Impossibile eliminare il file %s", $file);

}
```

Lo script termina stampando a video un errore se il file non esiste, altrimenti prosegue con l'operazione di cancellazione vera e propria.

Ottenere informazioni

I file contengono diverse **informazioni** riguardo se stessi, ad esempio la dimensione, la data di ultima modifica, l'id del proprietario.

Possiamo **recuperare queste informazioni** tramite la funzione stat() che prende il nome del file come parametro e restituisce un array contenente statistiche e informazioni sul file, elencate nella seguente tabella

Indice numerico	Indice associativo	Informazioni
0	dev	Numero del dispositivo
1	ino	Numero dell'inode
2	mode	Modalità di protezione dell'inode
3	nlink	Numero di collegamenti
4	uid	ID utente del proprietario
5	gid	ID di gruppo del proprietario
6	rdev	Tipo del dispositivo, se dispositivo inode
7	size	Dimensioni in byte
8	atime	Data dell'ultimo accesso
9	mtime	Data dell'ultima modifica
10	ctime	Data dell'ultimo cambiamento
11	blksize	Dimensioni dei blocchi per l'I/O del file system
12	blocks	Numero di blocchi allocati

Un esempio di output della funzione stat() eseguita su sistema operativo Linux

```
<?php

$stats = stat('robots.txt');

print_r($stats);

/* Output

Array

(
```

```
    [0] => 2050

    [1] => 6951840

    [2] => 33204

    [3] => 1

    [4] => 1004

    [5] => 1005

    [6] => 0

    [7] => 25

    [8] => 1572203082

    [9] => 1568766637

    [10] => 1572203082

    [11] => 4096

    [12] => 8

    [dev] => 2050

    [ino] => 6951840

    [mode] => 33204

    [nlink] => 1

    [uid] => 1004

    [gid] => 1005
```

```
    [rdev] => 0

    [size] => 25

    [atime] => 1572203082

    [mtime] => 1568766637

    [ctime] => 1572203082

    [blksize] => 4096

    [blocks] => 8

)

*/
```

Nota: non tutte le informazioni richiamabili tramite la funzione stat() sono disponibili in tutti i sistemi operativi. Ad esempio, in Windows non saranno disponibili l'ID utente e l'ID di gruppo perché, a differenza di Linux, Windows non fornisce tali informazioni.

Esistono anche alcune funzioni specifiche per ricavare le proprietà correlate al tempo, che restituiscono valori nel formato Unix timestamp:

- fileatime() restituisce l'ora in cui il file ha ricevuto l'ultimo accesso, o false in caso di errore.
- filectime() restituisce l'ora in cui il file è stato cambiato l'ultima volta, o false in caso di errore. Un file viene considerato come cambiato se viene creato o scritto o quando sono stati cambiati i suoi permessi.
- filemtime() restituisce l'ora in cui il file è stato modificato l'ultima volta, o false in caso di errore. Un file viene considerato come modificato se viene creato o i suoi contenuti sono cambiati.

La funzione filemtime() in particola si rivela molto utile per forzare l'aggiornamento di un foglio di stile di un sito web che ha subito una modifica recente.

Infatti, nella realizzazione di un sito web si è soliti memorizzare nella cache del browser risorse statiche come CSS, JS in maniera tale che i successivi accessi al sito siano più veloci. Tuttavia una eventuale modifica a uno di questi file statici non verrebbe visualizzata dall'utente che ha già visitato quel sito.

La soluzione è quella di modificare il nome del file per forzare l'aggiornamento della cache. Per fare ciò possiamo sfruttare filemtime() in questo modo

```
<link rel="stylesheet" href="/assets/css/amdesk.css?ver=<?php echo filemtime('/assets/css/amdesk.css')">
```

che produce un output simile a questo

```
<link rel="stylesheet" href="/assets/css/amdesk.css?ver=1574506234">
```

Leggere uno o più caratteri

Usata in combinazione con la funzione feof() che restituisce true quando viene raggiunta la fine di un file, la funzione fgetc() può essere utilizzata per **leggere un carattere alla volta** da un file.

La funzione accetta come argomento il puntatore al file e restituisce un solo carattere dal file a cui fa riferimento.

Di seguito è mostrato un esempio di funzionamento

```
<?php

$file = 'file.txt';

$handler = fopen($file, 'r');

while (!feof($handler)) {

    $char = fgetc($handler);

    echo $char."\n";

}

fclose($handler);
```

La funzione fgetc() é tanto semplice da usare quanto scomoda se abbiamo a che fare con file di grandi dimensioni, poiché leggendo un carattere per volta, è necessario un tempo molto lungo per leggere tutto il file.

Per fortuna PHP mette a disposizione la funzione fgets(), che consente di **leggere intere righe di caratteri**.

La funzione fgets() accetta due parametri: il puntatore al file e il parametro $lenght, restituendo una stringa di lunghezza massima in byte pari a $lenght - 1.

La lettura termina quando:

- È stato letto il numero specificato di byte.
- Viene incontrata una nuova riga.
- Viene raggiunta la fine del file.

```php
<?php

$handler = fopen("file.txt", "r");

if (false !== $handler) {

    while (false !== ($buffer = fgets($handler, 4096))) {

        echo $buffer;

    }

    if (!feof($handler)) {

        echo "Errore nella lettura tramite fgets()\n";

    }

    fclose($handler);

}
```

Accesso casuale ai dati

Le funzioni viste fino ad ora consentono di **leggere i dati in modo sequenziale**, cioè nello stesso ordine in cui sono disposti nel file.

Questa modalità di lettura crea una limitazione: quando l'**indicatore di posizione del file** supera un determinato punto, per tornare a leggere prima di quel punto è necessario chiudere e riaprire il file.

PHP offre la possibilità di **posizionarsi in un punto qualsiasi all'interno di un file** tramite la funzione fseek(), la cui sintassi è la seguente

fseek (resource $handle , int $offset [, int $whence = SEEK_SET]) : int

dove:

1. $handle è il puntatore al file
2. $offset è la nuova posizione corrente (in byte)
3. $whence è la posizione da cui iniziare a leggere

La funzioni restituisce 0 in caso di successo, -1 altrimenti. I possibili valori di $whence sono rappresentati da 3 costanti

1. SEEK_SET: l'inizio del file (default)
2. SEEK_CUR: posizione corrente
3. SEEK_END: fine del file

Nota: la funzione fseek() è comoda nei casi in cui sappiamo a priori come è stato costruito il file che dobbiamo leggere (ad esempio con le righe aventi lunghezza 10 caratteri).

Facciamo un esempio per chiarire meglio il funzionamento di fseek()

```
<?php

$filename = 'file.txt';

$offset = 35;

$handler = fopen($filename, 'w+');

if (false !== $handler) {

    for ($i = 1; $i < 10; $i++) {

        fwrite($handler, "Riga $i\n");

    }
```

```
    fseek($handler, $offset);

    $buffer = fread($handler, 1024);

}

fclose($handler);

echo $buffer;
```

Tramite un **ciclo for** scriviamo 9 volte una stringa di lunghezza fissa all'interno di un file, successivamente spostiamo l'indicatore di posizione avanti di 35 byte (riga 11), dunque la lettura successiva (riga 12) tramite fread() comincerà da "Riga 6" fino al termine.

Funzioni per gestire directory

Vediamo ora un set di **funzioni per manipolare le directory**. Alcune di queste utilizzano un descrittore di directory, mentre altre utilizzano una stringa che rappresenta il nome della directory.

Un **descrittore di directory** è molto simile a un descrittore di file, ossia un numero intero che punta a una directory.

Creare una cartella

La prima funzione che andiamo a presentare è mkdir() che consente di **creare una cartella**, specificandone il nome come parametro

```
$dirname = 'php';

if (false === mkdir($dirname)) {

    printf('Impossibile creare la directory %s', $dirname);

}
```

Se l'operazione è andata a buon fine, la funzione restituisce true.

Cancellare una cartella

Così come è possibile creare una directory, allo stesso modo la si può eliminare.

Per **cancellare una directory** dal file system è necessario utilizzare la funzione rmdir()

```
<?php

$dirname = 'php';

if (false === rmdir($dirname)) {

    printf('Impossibile eliminare la directory %s', $dirname);

}
```

Leggere il contenuto di una cartella

Per **leggere il contenuto di una cartella** è necessario inanzitutto farsi ritornare il puntatore alla directory. La funzione preposta a tale operazione è opendir().

Una volta che abbiamo il puntatore alla directory, possiamo utilizzare la funzione readdir() per leggere, un file alla volta, il contenuto della directory.

L'elenco delle voci ritornate da readdir() include pure . (che specifica la directory corrente) e .. (che specifica il genitore della directory corrente).

Alla fine dell'operazione di lettura possiamo rilasciare le risorse tramite la funzione closedir().

```
<?php

$handler = opendir('php-7');



if (false !== $handler) {

    while ($file = readdir($handler)) {
```

```
        echo $file."\n";

    }

}

closedir($handler);
```

Rinominare una cartella

Concludiamo questa guida con la funzione per **rinominare una cartella** che poi è la stessa utilizzata per rinominare i file, ossia rename()

```
<?php

$dirname = 'php-5';

if (false === rename($dirname, 'php-7')) {

    printf('Impossibile rinominare la directory %s', $dirname);

}
```

La lista completa sulle funzioni per gestire il file system è disponibile a **questo indirizzo**.

Nelle prossime guide vedremo invece come gestire i file **CSV**, **JSON** e **XML**.

File CSV: le funzioni PHP per gestirli

Nelle guide precedenti dedicate alla **gestione di file** abbiamo analizzato le **funzioni** che ci consentono di manipolare semplici file di testo.

Capita spesso però di dover gestire tipologie di file differenti da un file .txt, tra cui: file CSV, **file JSON**, file XML.

In questa guida ci concentriamo sulle funzioni che il linguaggio offre per **leggere e scrivere dati su file CSV**.

Indice dei contenuti

1. **Cosa sono i file CSV**
2. **Creare un file CSV**
3. **Leggere un file CSV**
4. **Leggere una stringa CSV**

Cosa sono i file CSV

L'acronimo **CSV** sta per *Comma-Separated Values* e, come dice il nome stesso è un formato di file dove i dati sono un **elenco di valori separati da una virgola**. Viene utilizzato per importare ed esportare tabelle di dati.

In questo formato, ogni **riga della tabella** è rappresentata da una linea di testo, che a sua volta è divisa in campi (le singole colonne) separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore.

La seguente tabella di esempio

Nome	Cognome	Città
Mario	Rossi	Roma
John	O'Really	New York
Angelica	Ramos	London

può essere rappresentata in formato CSV in questo modo

```
Mario;Rossi;Roma
John;O'Really;New York
Angelica;Ramos;London
```

In generale, in un file CSV si dovrebbero rispettare le seguenti regole:

1. Separare i campi tramite punto e virgola.
2. Se i campi contengono virgole o punto e virgole racchiuderli tra doppi apici.
3. Separare le righe con un a capo (\n);
4. Non lasciare spazi prima e dopo i campi.
5. Se il campo presenta un carattere di doppio apice racchiuderlo tra doppi apici: " diventa "".

I file CSV hanno estensione .csv e vengono utilizzati spesso nei **fogli di calcolo elettronici** come ad esempio Microsoft Excel o LibreOffice Calc.

Per quanto riguarda il separatore non esiste uno standard comune: nei paesi anglosassoni viene utilizzata la virgola, mentre in Italia e altri paesi latini il punto e virgola.

Creare un file CSV

Per **creare o scrivere un file CSV** in PHP è possibile utilizzare la funzione fputcsv() che prende come argomenti il puntatore al file e un **array di stringhe** da scrivere nel file.

La funzione restituisce il numero di byte scritti oppure false in caso di errore.

Di seguito è mostrato un esempio di utilizzo

```
<?php

$filename = 'users.csv';

$users = [

    ['Cristiano', 'Ronaldo', 'Portogallo'],

    ['Gonzalo', 'Higuain', 'Argentina'],

    ['Giorgio', 'Chiellini', 'Italia']

];

$handler = fopen($filename, 'w');

foreach($users as $user) {

    fputcsv($handler, $user, ';');

}

fclose($handler);
```

Tramite un **foreach** scriviamo nel file tutti gli elementi dell'array $users, separando i valori tramite il punto e virgola (riga 12).

È possibile specificare un terzo parametro (opzionale) della funzione, ossia il carattere delimitatore. Di default è la virgola, mentre nell'esempio abbiamo utilizzato il punto e virgola.

Leggere un file CSV

Leggere un file CSV è altrettanto semplice. PHP mette a disposizione la funzione fgetcsv() che preso un puntatore al file restituisce una riga letta dal file in questione oppure false.

Tramite un **ciclo while** possiamo quindi **leggere l'intero contenuto di un file CSV**. Vediamolo con un esempio

```
<?php
$filename = 'users.csv';
$handler = fopen($filename, 'r');


while($data = fgetcsv($handler)) {


    $users[] = implode('', $data);


}


fclose($handler);


print_r($users);


/* Output


Array


(


    [0] => Cristiano;Ronaldo;Portogallo


    [1] => Gonzalo;Higuain;Argentina


    [2] => Giorgio;Chiellini;Italia


)


*/
```

Leggere una stringa CSV

PHP consente di leggere anche una singola stringa CSV tramite la funzione str_getcsv()

```
<?php

$values = '10,89,100,34,15';

$csv = str_getcsv($values);

print_r($csv);

/* Output

Array

(

    [0] => 10

    [1] => 89

    [2] => 100

    [3] => 34

    [4] => 15

)

*/
```

Nota: nell'esempio precedente avremmo ottenuto lo stesso risultato utilizzando la **funzione di stringa explode()** che restituisce un array contenente i pezzi della stringa esplosi tramite il carattere separatore.

File JSON: cosa sono e come gestirli in PHP

Esistono diversi formati con cui le applicazioni client-server possono scambiarsi dati, uno dei quali è diventato ormai uno standard de facto: stiamo parlando di **JSON**.

In questa guida presentiamo **JSON e la sua sintassi e vediamo come codificare e decodificare dati in formato JSON** tramite le **funzioni** che PHP mette a disposizione.

Indice dei contenuti

1. **Cos'è JSON**
2. **Utilizzi**
3. **Sintassi**
4. **Codificare dati JSON**
5. **Decodificare dati JSON in oggetto PHP**
6. **Decodificare dati JSON in array PHP**
7. **Stampare dati JSON in tabella HTML**
8. **Restituire errori JSON**

Cos'è JSON

JSON è l'acronimo di **JavaScript Object Notation**, uno standard usato per l'interscambio di dati tra client e server e basato sulla rappresentazione degli **array associativi** di Javascript.

JSON, come XML, è un **formato basato su testo semplice** da comprendere sia per l'uomo che per i computer, ma a differenza di XML, le strutture di dati JSON sono più leggere e dunque necessitano di meno banda per il trasferimento.

I dati JSON vengono memorizzati su un **file** di testo che ha l'estensione `.json` e vanno inviati a chi ne fa richiesta, includendo nella risposta un'intestazione HTTP uguale a `Content-Type: application/json`.

Utilizzi

Ecco i possibili scenari in cui viene utilizzato il formato JSON:

- **web API**: per integrare un servizio web nelle proprie applicazioni, gli sviluppatori fanno uso delle API, ossia un'interfaccia per accedere in maniera lecita ai dati di quel servizio. Ad esempio api-football.com mette a disposizione le API per l'accesso a numerose statistiche sul calcio.
- **dati strutturati**: usati per migliorare l'esperienza d'uso nelle pagine dei risultati di ricerca, aiutano gli spider dei motori a comprendere meglio i contenuti di una pagina web. Il formato più utilizzato per implementarli è lo standard **JSON-LD**, basato appunto su JSON.
- **file di configurazione**: per funzionare correttamente, alcuni tool di sviluppo leggono le impostazioni da un file in formato JSON. È il caso del gestore di pacchetti Composer, che consente tramite un file `.json` di gestire le dipendenze di un progetto PHP.

Sintassi

La **sintassi JSON** è costituita da una serie di coppie chiave/valore

```
{ "key": "value" }
```

Le chiavi sono sempre di **tipo stringa**, mentre il valore della chiave può essere una stringa, un valore **booleano**, un **numero**, un valore nullo, un **array** o un oggetto.

Le stringhe devono essere racchiuse tra doppi apici e possono contenere caratteri di escape come \n, \t e \.

Il contenuto tipico di un file JSON assomiglia a questo:

```
{

    "player": "Cristiano Ronaldo",

    "team": "Juventus",

    "age": 34,

    "statistics": [

        {

            "matches": 989,

            "goals": 712,

            "assists": 219

        }

    ]

}
```

Nonostante JSON assomigli a **un oggetto** o un array associativo, in realtà è una **stringa serializzata**, il che significa che può essere decodificata in diversi **tipi di dati**.

Il **linguaggio PHP** consente di gestire questa tipologia di struttura tramite le seguenti funzioni:

- json_encode(): codifica un array associativo in JSON.
- json_decode(): decodifica un file JSON in un oggetto o array associativo.

- json_last_error() e json_last_error_message: restituiscono rispettivamente in formato intero e stringa, l'ultimo errore relativo alla codifica/decodifica con le funzioni elencate sopra.

Entrambe le funzioni json_encode() e json_decode() funzionano solo con dati di tipo stringa con codifica UTF-8.

Codificare dati JSON

In PHP la funzione json_encode() viene utilizzata per codificare un valore nel formato JSON. Il valore codificato può essere di qualunque tipo di dati tranne una risorsa, ad esempio un gestore di file.

L'esempio seguente mostra **come codificare un array associativo PHP in un oggetto JSON**:

```
<?php

$player = [

    "player" => "Cristiano Ronaldo",

    "age" => 34,

    "statistics" => [

        "matches" => 989,

        "goals" => 712,

        "assists" => 219

    ]

];

$json = json_encode($player);

var_dump($json);
```

L'ouput del programma è il seguente

```
string(94) "{"player":"Cristiano Ronaldo","age":34,"statistics":{"matches":989,"goals":712,"assists":219}}"
```

Se vogliamo salvare il contenuto della **variabile PHP** $json in un file JSON possiamo utilizzare la funzione file_put_contents() in questo modo

```
file_put_contents('player.json', $json);
```

Decodificare dati JSON in oggetto PHP

Decodificare dati JSON è altrettanto semplice quanto codificarli. Il linguaggio mette a disposizione la funzione json_decode() per convertire una stringa JSON nel tipo di dati PHP appropriato.

L'esempio seguente mostra **come decodificare o convertire un oggetto JSON in un oggetto PHP**

```
<?php

$data = '{

    "name": "Mario",

    "surname": "Rossi",

    "age": 40,

    "contacts": [

        {

            "email": "mariorossi@gmail.com",

            "phone": "3200202020"

        }

    ]

}';

$json = json_decode($data);
```

```
var_dump($json);
```

Nel listato precedente abbiamo memorizzato alcuni dati in formato JSON nella variabile $data. In alternativa, avremmo potuto salvarli in un file con estensione .json e richiamare tale file tramite la **funzione per leggere file** file_get_contents().

L'output del programma è il seguente

```
object(stdClass)#1 (4) {

  ["name"]=>

  string(5) "Mario"

  ["surname"]=>

  string(5) "Rossi"

  ["age"]=>

  int(40)

  ["contacts"]=>

  array(1) {

    [0]=>

    object(stdClass)#2 (2) {

      ["email"]=>

      string(20) "mariorossi@gmail.com"

      ["phone"]=>

      string(10) "3200202020"
```

```
    }


  }
}
```

Come possiamo notare, la funzione `json_encode()` restituisce un oggetto della classe `stdClass` coi valori codificati a seconda del tipo di dato trovato nella stringa JSON memorizzata in `$data`.

Possiamo anche accedere al valore di una singola chiave in questo modo

```
<?php

echo $json->name; // Mario

echo $json->age; // 40

echo $json->contacts[0]->email; // mariorossi@gmail.com
```

Decodificare dati JSON in array PHP

Di default, la funzione `json_decode()` restituisce un oggetto `stdClass`. Tuttavia, è possibile specificare opzionalmente un secondo parametro che accetta un valore booleano che, quando impostato a `true`, consente la **conversione dell'oggetto JSON in un array associativo**.

Di seguito è mostrato un esempio di utilizzo

```
<?php

$data = '{

    "name": "Mario",

    "surname": "Rossi",

    "age": 40,

    "contacts": [
```

```
        {

            "email": "mariorossi@gmail.com",

            "phone": "3200202020"

        }

    ]

}';

$json = json_decode($data, true); // il secondo parametro settato a true specifica che vogliamo farci restituire un
array associativo

var_dump($json);
```

Il codice visto in precedenza produrrà il seguente output

```
array(4) {

  ["name"]=>

  string(5) "Mario"

  ["surname"]=>

  string(5) "Rossi"

  ["age"]=>

  int(40)

  ["contacts"]=>

  array(1) {
```

```
    [0]=>

    array(2) {

      ["email"]=>

      string(20) "mariorossi@gmail.com"

      ["phone"]=>

      string(10) "3200202020"

    }

  }

}
```

È dunque possibile accedere ai dati come in un normale array associativo.

Stampare dati JSON in tabella HTML

Per **formattare i dati di un file JSON** in formato tabellare possiamo utilizzare un **ciclo foreach** per iterare l'oggetto che ci viene restituito dalla funzione json_decode().

Poniamo di avere i dati memorizzati in un file chiamato users.json. Il contenuto di tale file è il seguente

```
[{"name":"Pinco","surname":"Pallino"},{"name":"Mario","surname":"Rossi"},{"name":"Tizio","surname":"Gialli"},{"name
":"Sempronio","surname":"Marroni"}]
```

mentre il codice PHP che mostra i dati in una tabella è il seguente

```
<?php

$file = file_get_contents("users.json");

$users = json_decode($file);
```

```
if (JSON_ERROR_NONE !== json_last_error()) {

    printf("%s", json_last_error_msg());

    exit;

}

?>

<table>

    <tbody>

        <?php foreach ($users as $user):?>

        <tr>

            <td><?php echo $user->name?></td>

            <td><?php echo $user->surname?></td>

        </tr>

        <?php endforeach;?>

    </tbody>

<table>
```

che, salvo errori, produce il seguente output HTML

```
<table>

    <tbody>
```

```
        <tr>

            <td>Pinco</td>

            <td>Pallino</td>

        </tr>

        <tr>

            <td>Mario</td>

            <td>Rossi</td>

        </tr>

        <tr>

            <td>Tizio</td>

            <td>Gialli</td>

        </tr>

    </tbody>

<table>
```

Restituire errori JSON

La funzione json_last_error() dell'esempio precedente restituisce un numero intero equivalente ad una costante che può riferirsi ad uno specifico errore.

Se ad esempio la funzione restituisce il numero 4, ciò indica un **errore di sintassi JSON**, come mostrato dalla funzione json_last_error_msg(), la quale mostra l'errore in formato stringa.

Per un elenco completo delle costanti di errore della funzione json_last_error() è possibile **consultare la documentazione**.

File XML: cosa sono e come gestirli in PHP

L'ultimo formato di **file** che andiamo a presentare è l'**eXtensible Markup Language**, meglio noto come **XML**. Si tratta di uno standard che consente di **formattare e memorizzare** in maniera persistente dati strutturati.

È un formato molto conosciuto nel mondo informatico, tuttavia con l'introduzione della fattura elettronica è probabilmente diventato noto anche ai non addetti ai lavori.

PHP offre un'ampia gamma di **funzioni** per gestire i **documenti in formato XML**, qui vedremo solamente alcune di queste in quanto non basterebbe una semplice guida per analizzarle tutte in dettaglio.

Indice dei contenuti

1. **Cos'è XML**
2. **Sintassi di un file XML**
3. **Leggere formato XML in PHP**
4. **Accedere ai singoli valori**
5. **Leggere file XML in PHP**

Cos'è XML

XML è una specifica nata per creare **linguaggi di markup** (marcatura), ossia linguaggi che consentono di descrivere i dati attraverso l'utilizzo di elementi chiamati *tag*. XML é molto simile ad HTML come struttura, tuttavia esistono due sostanziali differenze con il linguaggio base utilizzato per creare siti web:

- XML non definisce un insieme specifico di tag che è necessario utilizzare: possiamo usare tag personalizzati (da qui l'acronimo eXtensible)
- Un file XML non ammette errori di sintassi: mentre una pagina HTML funziona anche se omettiamo la chiusura di un tag, la lettura di un documento XML mal formattato genererebbe un errore.

Un **documento XML** ha estensione .xml e il suo contenuto è leggibile attraverso un qualsiasi editor di testo semplice. Vediamo un esempio di file XML e analizziamone la sintassi.

Sintassi di un file XML

Come già detto, un file XML è composto da una serie di **tag** che descrivono il contenuto del documento. Questi vengono indicati tramite i caratteri < e >.

Un esempio di file XML è il seguente

```
<?xml version="1.0" encoding="UTF-8"?>

<cars>

    <car>
```

```
        <brand>Renault</brand>

        <model>Clio</model>

        <engine>2.0</engine>

    </car>

</cars>
```

- La **riga 1** contiene informazioni circa la versione e la codifica utilizzati per il documento. In questo caso la codifica è utf-8.
- Le **righe 2-8** contengono gli elementi, ciascuno dei quali ha un marcatore di inizio e uno di fine, quest'ultimo indicato con l'aggiunta del carattere / (slash).
- Il testo delimitato dal tag di apertura e da quello di chiusura rappresenta il valore di un elemento.
- Il primo elemento (cars) è chiamato elemento radice o root. Insieme agli elementi successivi forma una struttura ad albero.

Abbiamo dato dei nomi personalizzati ai tag. In HTML invece esistono un set di tag da utilizzare per formattare il documento, ad esempio con i tag

```
<p></p>
```

realizziamo un paragrafo.

PHP mette a disposizione diverse **funzioni per la gestione dei file XML**. Vediamo alcune di queste.

Leggere formato XML in PHP

Per leggere un documento in formato XML in PHP è possibile utilizzare l'estensione SimpleXML che, come dice il nome stesso, consente una gestione semplice di questo formato.

Esistono due funzioni che possiamo utilizzare a tale scopo:

- simplexml_load_file(): legge un file in formato XML e restituisce un oggetto della classe SimpleXMLElement.
- simplexml_load_string(): legge una stringa in formato XML e restituisce un oggetto della classe SimpleXMLElement.

Entrambe le funzioni restituiscono false in caso di errore.

> **Nota:** Dato che entrambe le funzioni possono restituire valori non booleani che vengono valutati come false, è bene ricordarsi di usare l'operatore di confronto === nelle **condizioni**.

Prendendo l'esempio precedente, possiamo scrivere il seguente codice PHP

```php
<?php

$xml = <<<XML

<?xml version="1.0" encoding="UTF-8"?>

<cars>

    <car>

        <brand>Renault</brand>

        <model>Clio</model>

        <engine>2.0</engine>

    </car>

</cars>

XML;

var_dump(simplexml_load_string($xml));
```

che produce l'output

```
object(SimpleXMLElement)#1 (1) {

  ["car"]=>

  object(SimpleXMLElement)#2 (3) {

    ["brand"]=>

    string(7) "Renault"
```

```
    ["model"]=>

    string(4) "Clio"

    ["engine"]=>

    string(3) "2.0"

  }

}
```

Nel nostro esempio l'oggetto ritornato rappresenta l'elemento cars del documento e ha una proprietà di nome car che contiene a sua volta un altro oggetto SimpleXMLElement, contenente 3 proprietà: brand, model e engine.

Vediamo come è possibile accedere in maniera singola a ciascuno di questi elementi.

Accedere ai singoli valori

Come un qualsiasi altro oggetto, tutte le proprietà di un oggetto della classe SimpleXMLElement sono accessibili tramite l'operatore ->.

Vediamo alcuni esempi

```
<?php

$xml = <<<XML

<?xml version="1.0" encoding="UTF-8"?>

<cars>

    <car>

        <brand>Renault</brand>

        <model>Clio</model>

        <engine>2.0</engine>
```

```
    </car>

</cars>

XML;

$simpleXML = simplexml_load_string($xml);

echo "Marca ".$simpleXML->car->brand."\n"; // Renault

echo "Modello ".$simpleXML->car->model."\n"; // Clio

echo "Cilindrata ".$simpleXML->car->engine."\n"; // 2.0
```

Negli esempi mostrati fino ad ora abbiamo utilizzato la sintassi *Heredoc* tramite il <<<, che ci consente di gestire stringhe su più righe senza la necessità di effettuare concatenazione e migliorando la leggibilità.

Leggere file XML in PHP

Vediamo adesso un esempio di come leggere un file XML tramite la funzione simplexml_load_file().

Supponiamo di avere un file cars.xml con il seguente contenuto

```
<?xml version="1.0" encoding="UTF-8"?>

<cars>

    <car>

        <brand>Ferrari</brand>

        <model>F40</model>

        <engine>2.9 V8</engine>

    </car>

    <car>
```

```
        <brand>Porsche</brand>

        <model>Carrera GT</model>

        <engine>5.7 V10</engine>

    </car>

    <car>

        <brand>Lamborghini</brand>

        <model>Diablo</model>

        <engine>5.7 V12</engine>

    </car>

    <car>

        <brand>BMW</brand>

        <model>M3</model>

        <engine>3.2 L6</engine>

    </car>

</cars>
```

Vogliamo leggere il file con PHP e mostrare il risultato in una tabella HTML.

Di seguito è mostrato il codice necessario

```
<?php

$cars = simplexml_load_file('cars.xml');

?>

<table>

    <tbody>

        <?php foreach ($cars as $car):?>

        <tr>

            <td><?php echo $car->brand;?></td>

            <td><?php echo $car->model;?></td>

            <td><?php echo $car->engine;?></td>

        </tr>

        <?php endforeach;?>

    </tbody>

<table>
```

che dà come output il seguente codice HTML

```
<table>

    <tbody>

        <tr>
```

```
            <td>Ferrari</td>
            <td>F40</td>
            <td>2.9 V8</td>
        </tr>


        <tr>


            <td>Porsche</td>


            <td>Carrera GT</td>


            <td>5.7 V10</td>


        </tr>


        <tr>


            <td>Lamborghini</td>


            <td>Diablo</td>


            <td>5.7 V12</td>


        </tr>


        <tr>


            <td>BMW</td>


            <td>M3</td>


            <td>3.2 L6</td>


        </tr>


    </tbody>


<table>
```

Introduzione ai database e MySQL

Nelle guide precedenti abbiamo visto come memorizzare in maniera permanente informazioni tramite i **file** e le **funzioni** che PHP mette a disposizione per leggere e scrivere dati su diversi **formati di file**.

In molti scenari i file sono in grado di svolgere il loro compito in maniera egregia, ma quando dobbiamo gestire una gran mole di dati sorgono problemi dovuti ai limiti imposti dai file, come l'impossibilità di accedere al contenuto di un file da parte di più utenti contemporaneamente oppure l'eccessivo uso di risorse nel caso di file di grosse dimensioni (nell'ordine dei GigaByte).

È in questo scenario che entrano in gioco i database. In questa guida faremo una breve **introduzione ai database e vedremo cos'è MySQL**, i suoi vantaggi e cos'è il linguaggio SQL. Nelle guide successive invece vedremo come utilizzare MySQL per creare applicazioni PHP potenti, efficienti e in grado di gestire una gran quantità di informazioni.

ndice dei contenuti

1. **Cos'è un database**
2. **DBMS e MySQL**
3. **Come funziona MySQL**
4. **Il linguaggio SQL**
5. **Creare un database**
6. **Creare un tabella**
7. **Inserire i dati**
8. **Modificare i dati**
9. **Selezionare i dati**
10. **PHP e MySQL**

Cos'è un database

Un **database**, letteralmente **base di dati**, è una raccolta di dati organizzata in modo ordinato, così come una biblioteca può essere considerata una raccolta di libri.

Esistono diversi tipi di database: i primi ad essere utilizzati sono stati i **database relazionali**, seguiti dai database orientati agli oggetti nei primi anni 90 e dai database NoSQL che sono nati per soddisfare l'esigenza di maggiore velocità ed elaborazione dei dati non strutturati.

Il simbolo del database

Il modello più utilizzato ancora oggi è quello relazionale. Gli elementi di un database relazionale sono organizzati sotto forma di **tabelle composte da colonne e righe**. Questo tipo di database offre la soluzione più efficiente e flessibile per accedere alle informazioni.

DBMS e MySQL

Un database viene gestito tramite il **DBMS (Database Management System)**, letteralmente sistema di gestione di database, ossia un software che consente il controllo e la gestione completa di un database.

Nella pratica, un DBMS funge da **interfaccia tra il database e gli utenti finali** consentendo tra le altre cose la creazione di database, la memorizzazione, l'interrogazione, la modifica e l'eliminazione dei dati, oltre alla gestione dell'integrità degli stessi.

Inoltre, un DBMS consente la gestione amministrativa di un database, offrendo operazioni quali monitoraggio delle performance, backup e ripristino dei dati.

La parola database viene utilizzata sia per riferirsi al database vero e proprio, sia per riferirsi al DBMS, dunque d'ora in avanti ci riferiremo al DBMS come al database.

PHP è in grado di gestire un gran numero di database, alcuni di questi sono: SQLite, MySQL, PostgreSQL, Oracle, Microsoft Access, Microsoft SQL Server, MongoDB. In generale è possibile utilizzare uno qualsiasi di questi database nelle nostre applicazioni PHP. Tuttavia in questa e nelle guide successive ci concentreremo su MySQL.

MySQL è uno dei più popolari database attualmente in uso sul web ed è senza dubbio il più popolare utilizzato in combinazione col **linguaggio PHP**. È open source ed è disponibile nel pacchetto di installazione di **XAMPP**.

MySQL offre numerosi vantaggi:

- è stato **progettato e ottimizzato** per le applicazioni web, comprese quelle sviluppate in PHP
- è **facile da usare**, ma estremamente potente, veloce, sicuro e scalabile
- supporta **SQL** (Structured Query Language), il linguaggio utilizzato per interrogare i database
- è **multipiattaforma** e funziona su diversi sistemi operativi inclusi Linux, Windows e Mac OS
- PHP e MySQL sono **la migliore accoppiata** per lo sviluppo di applicazioni web di piccole e grandi dimensioni
- include **strumenti di amministrazione** intuitivi e potenti come phpMyAdmin o MySQL Workbench.

Come funziona MySQL

MySQL archivia i dati in tabelle come altri database relazionali. Una **tabella è una raccolta di dati correlati** ed è divisa in **righe** e **colonne**.

Ciascuna riga in una tabella rappresenta un set di dati intrinsecamente connessi tra loro, mentre ogni colonna rappresenta un campo specifico.

Ritornando all'esempio della biblioteca, un'ipotetica tabella di nome libri potrebbe essere rappresentata in questo modo

```
+-----+-------------------------+------+

|  id | titolo                  | anno |

+-----+-------------------------+------+

|  1  | Il signore degli anelli | 1955 |

|  2  | Shining                 | 1977 |

|  3  | Frankenstein            | 1818 |

|  4  | Il Gattopardo           | 1958 |

|  5  | La forma dell'acqua     | 1994 |

+-----+-------------------------+------+
```

Id, titolo e anno rappresentano le colonne della tabella, mentre ciascuna riga contenente i valori di ogni colonna è un record. Vediamo come è possibile prelevare questi dati per poterli utilizzare nelle nostre applicazioni. Introduciamo allo scopo un nuovo linguaggio.

Il linguaggio SQL

SQL, Structured Query Language, è il linguaggio utilizzato per comunicare con i database relazionali come MySQL. Con SQL è possibile eseguire qualsiasi operazione relativa ai database come la creazione di database e tabelle, il salvataggio dei dati l'aggiornamento e l'eliminazione dei dati e l'interrogazione di un database per specifici record.

Con SQL é possibile:

- creare un database
- inserire, modificare e gestire i dati
- selezionare i dati
- eliminare i dati
- gestire i permessi di accesso ai dati

I comandi utilizzati per effettuare tali operazioni vengono chiamati *query*. Nei paragrafi seguenti vedremo la sintassi SQL per ciascuna delle operazioni appena elencate.

Per approfondire rimandiamo alla lettura della **documentazione MySQL**.

Creare un database

Tramite l'istruzione CREATE DATABASE è possibile creare un nuovo database. È sufficiente indicare il nome del database dopo l'istruzione, come mostra il seguente esempio

```
CREATE DATABASE biblioteca
```

Creare una tabella

Una volta creata la struttura del database è necessario aggiungere le tabelle. È possibile farlo tramite il costrutto CREATE TABLE seguito dal nome della tabella e dai campi.

Vediamo un esempio di utilizzo

```
CREATE TABLE `libri` (

  `id` int(11) PRIMARY KEY NOT NULL,

  `titolo` varchar(80) NOT NULL,

  `anno` int(4) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
ALTER TABLE `libri`

CHANGE `id` `id` INT(11) NOT NULL AUTO_INCREMENT;
```

Con la query precedente abbiamo creato una tabella di nome libri con i seguenti campi:

- id di tipo integer
- titolo di tipo varchar
- anno di tipo integer

Inoltre abbiamo definito il campo id come chiave primaria e reso questo campo autoincrementante tramite la clausola AUTOINCREMENT contenuta nella query successiva ALTER TABLE, utilizzata per modificare la struttura di una tabella.

Inoltre ogni volta che viene inserito un nuovo record, il valore di id viene incrementato in maniera automatica di una unità prendendo in considerazione il valore dell'ultimo id inserito.

Inserire i dati

L'inserimento di un nuovo record in una tabella già esistente viene effettuato tramite l'istruzione INSERT

```
INSERT INTO libri (id, titolo, anno) VALUES (0, 'La storia infinita', 1979)
```

La INSERT prevede che la tabella che si vuole popolare esista e per ciascun campo venga rispettato il tipo.

> **Nota:** specificando il valore 0 per il campo id, MySQL provvederà ad incrementarne il valore a partire dall'ultimo id inserito.

Modificare i dati

L'istruzione UPDATE consente di modificare il contenuto di alcuni campi nei record che soddisfano una determinata condizione posta in WHERE. Se non si indica nessuna condizione vengono modificati tutti i record.

Con la clausola SET si indica il campo o i campi da modificare seguiti dal simbolo uguale e dal nuovo valore.

Vediamo un esempio di utilizzo per modificare un singolo campo

```
UPDATE libri SET anno = 1900 WHERE id = 4
```

e per modificare campi multipli

```
UPDATE libri SET anno = 1900, titolo = 'Nuovo titolo' WHERE id = 4
```

Selezionare i dati

L'istruzione SELECT consente di estrarre un elenco di record dalla tabella indicata nella clausola FROM e che soddisfano una determinata condizione (opzionale) impostata tramite l'operatore WHERE.

Di seguito è mostrato un esempio di query che restituisce tutti i titoli dalla tabella libri

```
SELECT titolo FROM libri
```

Se si esegue la query verranno restituiti i seguenti record

```
+-------------------------+
| titolo                  |
+-------------------------+
| Il signore degli anelli |
| Shining                 |
| Frankenstein            |
| Il Gattopardo           |
| La forma dell'acqua     |
+-------------------------+
```

Se vogliamo ottenere il titolo del libro il cui codice è uguale a 128489 dobbiamo eseguire la stessa query precedente con l'istruzione WHERE

```
SELECT titolo FROM libri WHERE id = 2
```

che restituirà

```
+-------------------------+
```

```
| titolo                  |

+-------------------------+

| Shining                 |

+-------------------------+
```

Eliminare i dati

L'eliminazione dei dati avviene tramite il costrutto DELETE ed è possibile:

1. Cancellare tutti i record della tabella
2. Cancellare un singolo record

Per cancellare un record possiamo utilizzare la seguente interrogazione SQL

```
DELETE FROM libri WHERE id = 2
```

Una successiva interrogazione alla tabella libri restituirà i seguenti record

```
+-------------------------+

| titolo                  |

+-------------------------+

| Il signore degli anelli |

| Frankenstein            |

| Il Gattopardo           |

| La forma dell'acqua     |

+-------------------------+
```

Per cancellare tutti i record è sufficiente omettere l'istruzione WHERE

PHP e MySQL

PHP mette a disposizione due **estensioni** per gestire un database MySQL:

1. L'estensione **MySQLi** che consente di accedere alle funzionalità fornite da MySQL 4.1 e versioni successive.
2. L'estensione **PDO** che fornisce un'interfaccia leggera per interagire con qualsiasi database incluso MySQL.

Al posto di usare una di queste due estensioni, nelle applicazioni moderne è ormai d'uso comune l'utilizzo di un ORM (Object Relational Mapping), una tecnica di programmazione che consente di mappare in classi i dati del database consentendo al programmatore di sfruttare la programmazione ad oggetti.

Gli ORM verranno discussi nella guida avanzata dedicata allo sviluppo e manutenzione di applicazioni web professionali. Nella guida successiva invece analizzeremo **MySQLi** e le sue funzioni base per l'accesso e la gestione di un database MySQL in PHP.

Estensione MySQLi: come utilizzarla per gestire database MYSQL

Per memorizzare o accedere ai dati archiviati all'interno di un **database MySQL** è necessario effettuare prima la connessione allo stesso. PHP consente di svolgere queste operazioni tramite due estensioni: MySQLi (MySQL improved) e **PDO** (PHP Data Objects).

Questa guida è dedicata a **MySQLi** e le sue funzioni, vedremo esempi di codice per creare un database ed effettuare le cosiddette **CRUD (Create, Read, Update, Delete)** ossia le 4 operazioni fondamentali che possiamo effettuare su una base dati.

Indice dei contenuti

1. **Cos'è MySQLi**
2. **Connessione al database**
3. **Chiusura della connessione**
4. **La funzione mysqli_query**
5. **Creazione database - CREATE DATABASE**
6. **Creazione tabelle - CREATE TABLE**
7. **Inserimento dati - INSERT INTO**
8. **Estrazione dati - SELECT**
9. **Aggiornamento dati - UPDATE**
10. **Cancellazione dati - DELETE**

Cos'è MySQLi

MySQLi (la i sta per improved, che significa migliorata) **è un'estensione PHP per la gestione di un database** MySQL. È stata sviluppata per sfruttare le nuove funzionalità presenti nella versione di **MySQL** 4.1.3 e successive. MySQLi è disponibile a partire da **PHP 5** e fornisce un modo semplice per connettersi ed eseguire query su un database MySQL.

L'estensione supporta sia il paradigma di programmazione procedurale che quello orientato agli oggetti. Poiché non abbiamo ancora affrontato la programmazione ad oggetti in questa guida useremo l'approccio di tipo **procedurale** che va bene per applicazioni semplici che non necessitano di continue modifiche.

Per approfondire **https://www.php.net/manual/en/mysqli.quickstart.dual-interface.php**

Connessione al database

È possibile effettuare la **connessione ad un database** MySQL usando la **funzione PHP** mysqli_connect(), la cui sintassi è la seguente

```
mysqli_connect($host, $username, $password, $dbname);
```

dove:

- $host è l'indirizzo della macchina dove è installato MySQL. Può essere il nome (in locale sarà "localhost") o l'indirizzo IP.
- $username è il nome dell'utente a cui sono stati concessi i privilegi per eseguire i **comandi SQL**.
- $password è la password associata all'utente $username.
- $dbname: è il nome del database con il quale vogliamo interagire.

La funzione restituisce false se la connessione non viene stabilita, altrimenti restituirà un **oggetto di tipo mysqli**. Una volta instaurata una connessione tutte le successive comunicazioni tra PHP e MySQL avverranno tramite questa connessione.

Supponiamo di aver installato MySQL in locale (sul nostro computer) e aver creato l'utente test con password test

> **Nota:** tra le opzioni disponibili per installare MySQL c'è **XAMPP**, un'ambiente di lavoro che oltre Apache e PHP include anche il DBMS.

Di seguito è mostrato un esempio di utilizzo della funzione mysqli_connect()

```
<?php

$conn = mysqli_connect("localhost", "test", "test");

if(false === $conn){

    exit("Errore: impossibile stabilire una connessione " . mysqli_connect_error());

}

echo "Connesso: " . mysqli_get_host_info($conn);
```

Se la connessione va a buon fine dovremmo ottenere il seguente output

```
Connesso a localhost via TCP/IP
```

> **Nota**: se si verifica un errore, la funzione mysqli_connect_error() restituisce un **messaggio di errore** relativo all'ultimo tentativo di connessione.

Chiusura della connessione

La connessione al database verrà chiusa automaticamente non appena termina l'esecuzione dello script. Tuttavia, una volta che abbiamo terminato di lavorare coi dati è buona prassi **chiudere la connessione al database** per rilasciare le risorse. È possibile farlo tramite la funzione mysqli_close().

Adesso vediamo come creare un database con relative tabelle e dati e come effettuare operazioni su di essi.

La funzione mysqli_query

La funzione mysqli_query() viene utilizzata per **eseguire query su un database**. La funzione accetta come parametri la connessione al database e una **query SQL**

mysql_query($conn, $query);

In caso di successo la funzione restituisce true se eseguiamo istruzioni di tipo CREATE, INSERT, DELETE, UPDATE, oppure un oggetto mysqli se eseguiamo una SELECT.

Vediamo adesso come utilizzare la funzione mysqli_query() per creare database e tabelle, inserire dati e operare su di essi.

Creazione database - CREATE DATABASE

Per **creare un nuovo database MySQL** abbiamo bisogno dell'istruzione SQL CREATE DATABASE, quindi dobbiamo eseguire tale istruzione passandola alla funzione mysqli_query() di PHP per creare il nostro database.

L'esempio seguente crea un database di nome *clienti*

```
<?php

$conn = mysqli_connect("localhost", "test", "test");

$sql = "CREATE DATABASE clienti";

if(false === mysqli_query($conn, $sql)) {

    echo "Errore: impossibile eseguire la query. " . mysqli_error($conn);

}

mysqli_close($conn);
```

Creazione tabelle - CREATE TABLE

Come già sappiamo dalla guida sull'**introduzione ai database**, i dati sono memorizzati in tabelle con un nome univoco, ciascuna delle quali è composta da righe che rappresentano i record e colonne che sono i campi della tabella.

Ora che abbiamo creato il nostro database non ci resta che creare le tabelle che memorizzeranno i dati. L'istruzione CREATE TABLE viene utilizzata per **creare una tabella nel database**. È possibile specificare i campi della tabella e il tipo di dato.

Vediamo un esempio di codice PHP per creare una tabella di nome *anagrafica* per il database *clienti*

```
<?php

$conn = mysqli_connect("localhost", "test", "test", "clienti");

$sql = "

CREATE TABLE anagrafica (
    id INT(11) NOT NULL AUTO_INCREMENT,
    nome VARCHAR(255) NOT NULL,
    cognome VARCHAR(255) NOT NULL,
    eta SMALLINT(3) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE = InnoDB CHARSET=utf8mb4 COLLATE utf8mb4_general_ci;

";

if(false === mysqli_query($conn, $sql)) {

    exit("Errore: impossibile eseguire la query. " . mysqli_error($conn));

}

echo "Tabella creata con successo";

mysqli_close($conn);
```

Il codice PHP nell'esempio precedente crea la tabella *anagrafica* contenente quattro campi: *id*, *nome*, *cognome*, *eta*.

Possiamo notare che ciascun campo è seguito da una dichiarazione del **tipo di dati**, la quale specifica quale tipo di dati può contenere la colonna. Il tipo VARCHAR indica un campo di **tipo stringa**, INT e SMALLINT di **tipo intero**.

Inoltre abbiamo aggiunto alcuni vincoli specificati dopo il nome della colonna. I vincoli definiscono le regole relative ai valori consentiti nelle colonne.

Ad esempio:

- PRIMARY KEY o chiave primaria è un campo che identifica i record della tabella in modo univoco.
- AUTOINCREMENT indica a MySQL che il campo deve incrementarsi di una unità dopo ogni nuovo inserimento.
- NOT NULL indica che il campo non può essere vuoto.

Per maggiori informazioni fare riferimento alla **documentazione MySQL**.

Adesso vediamo come è possibile popolare la tabella con alcuni dati.

Inserimento dati - INSERT INTO

L'istruzione INSERT INTO viene utilizzata per **inserire dati nella tabella MySQL**. Di seguito è mostrata la sintassi:

INSERT INTO nome_tabella (campo1, campo2, ... campoN) VALUES (valore1, valore2, ... valoreN);

Adesso facciamo un esempio concreto di codice PHP utilizzando l'istruzione INSERT INTO con i valori appropriati per la nostra tabella *anagrafica*, precedentemente creata.

Anche in questo caso ci serviremo della funzione mysqli_query() di PHP per inserire i dati nella tabella.

```
<?php

$conn = mysqli_connect("localhost", "test", "test", "clienti");

$sql = "INSERT INTO anagrafica (nome, cognome, eta) VALUES ('Mario', 'Rossi', 40)";

if(false === mysqli_query($conn, $sql)) {

    exit("Errore: impossibile eseguire la query. " . mysqli_error($conn));

}

echo "Dati inseriti con successo";

mysqli_close($conn);
```

Nell'esempio appena visto viene inserita una nuova riga nella tabella *anagrafica* specificando dei valori per i campi *nome*, *cognome* e *eta*. I valori per i campi *nome* e *cognome* vengono inseriti tra singoli apici in quanto il loro tipo è VARCHAR, mentre il campo *eta* viene inserito come numero intero.

> **Nota:** non abbiamo specificato nessun valore per il campo *id* in quanto è autoincrementante, ciò vuol dire che MySQL assegnerà automaticamente un valore a questo campo se non viene specificato, aumentando il valore precedente di 1.

Cosa succede se vogliamo inserire un valore contenente un apice? Possiamo racchiuderlo tra doppi apici oppure aggiungere la sequenza di escape \'.

Facciamo un esempio: se proviamo ad eseguire la query

```
$sql = "INSERT INTO anagrafica (nome, cognome, eta) VALUES ('John', 'O'Really', 40)";
```

otteniamo il seguente errore

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'Really', 40)' at line 1
```

poiché la stringa "O'Really" contiene un apice.

Facendo l'escape di questo carattere, ossia anteponendogli il carattere \ risolviamo

```
$sql = "INSERT INTO anagrafica (nome, cognome, eta) VALUES ('John', 'O\'Really', 40)";
```

In questo esempio abbiamo utilizzato una stringa predefinita, ma se il valore provenisse da un input utente, ossia da un modulo contatti dove l'utente può inserire qualsiasi cosa?

In generale l'approccio migliore è l'utilizzo delle ***Prepared Statement*** (letteralmente istruzioni preparate), una funzionalità integrata in MySQL 4.1 e versioni successive.

Tramite questa funzionalità è possibile inserire nel database in maniera sicura i dati prelevati da un modulo HTML, proteggendo l'applicazione da una vulnerabilità nota come **SQL Injection**. Affronteremo questo argomento in maniera più approfondita in una guida dedicata.

Estrazione dati - SELECT

L'istruzione SELECT di MySQL viene utilizzata per **selezionare i dati da una o più tabelle** del database.

Prima di vedere un esempio concreto in PHP, descriviamo brevemente i passaggi necessari per recuperare i dati da una tabella e stamparli a video:

- Creiamo una connessione al database usando mysqli_connect().
- Utilizziamo mysqli_query() per eseguire una SELECT sul database.
- In caso di successo la funzione mysqli_query() restituisce una risorsa di tipo mysqli_result.
- La risorsa restituita viene passata alla funzione mysqli_fetch_array().
- La funzione mysqli_fetch_array() restituisce un array di stringhe per ogni record della tabella oppure NULL se non ci sono record.

- Tramite un **ciclo while** stampiamo i dati di ciascuna record.
- Liberiamo le risorse con le funzioni mysqli_free_result() e chiudiamo la connessione.

```
<?php

$conn = mysqli_connect("localhost", "test", "test", "clienti");

$sql = "SELECT nome, cognome, eta FROM anagrafica";

$result = mysqli_query($conn, $sql);

if($result === false) {

    exit("Errore: impossibile eseguire la query. " . mysqli_error($conn));

}

while ($row = mysqli_fetch_array($result)) {

    echo $row['nome'].' '.$row['cognome'].' '.$row['eta'].'<br>';

}

mysqli_free_result($result);

mysqli_close($conn);
```

Nell'esempio precedente abbiamo selezionato i campi di tutti i record della tabella. Se volessimo estrarre solamente i record che rispondono a una precisa condizione dobbiamo aggiungere la clausola WHERE nella query.

Ad esempio la seguente query estrae tutti gli utenti con età maggiore di 40

```
SELECT nome, cognome

FROM anagrafica

WHERE eta > 40
```

Aggiornamento dati - UPDATE

Un'altra operazione utilizzata molto frequentementi sui record di un database è l'aggiornamento (o modifica).

L'istruzione UPDATE consente di **modificare i record esistenti in una tabella del database**. In genere è usata in combinazione con la clausola WHERE per applicare le modifiche solo ai record che soddisfano specifiche condizioni.

La sintassi è la seguente

UPDATE nome_tabella SET campo1 = valore1, campo2 = valore2, ... WHERE condizione;

Di regola:

- È possibile aggiornare uno o più campi con una sola query.
- È possibile specificare una o più condizioni tramite la clausola WHERE.
- È possibile aggiornare i valori di una tabella alla volta.

Consideriamo adesso i record della nostra tabella *anagrafica* di esempio

```
+----+------------+-----------+------+

| id | nome       | cognome   | eta  |

+----+------------+-----------+------+

|  1 | Mario      | Rossi     |  40  |

|  2 | John       | Smith     |  32  |

|  3 | Pippo      | Bianchi   |  35  |

+----+------------+-----------+------+
```

Supponendo di voler modificare il nome dell'utente con id uguale a 3, possiamo scrivere il seguente codice PHP

```
<?php

$conn = mysqli_connect("localhost", "test", "test", "clienti");

$sql = "UPDATE anagrafica SET nome = 'Carlo' WHERE id = 3";
```

```
$result = mysqli_query($conn, $sql);




if(mysqli_affected_rows($conn) > 0) {

    echo "Utente aggiornato con successo";

}

mysqli_close($conn);
```

In questo esempio abbiamo introdotto la funzione mysql_afftected_rows() che restituisce il numero di righe coinvolte nell'aggiornamento tramite UPDATE. Se il numero restituito è maggiore di zero, verrà eseguito l'echo del messaggio presente dentro la **condizione if**.

> **Nota:** mysql_affected_rows() può essere utilizzata dopo l'esecuzione delle seguenti istruzioni SQL: UPDATE, INSERT, REPLACE, DELETE.

Cancellazione dati - DELETE

Una volta popolata una tabella, è possibile **eliminare i record** utilizzando l'istruzione DELETE. Anche in questa caso, l'istruzione SQL viene utilizzata in combinazione con WHERE per eliminare solo i record che soddisfano una determinata condizione.

La sintassi di base dell'istruzione DELETE è la seguente:

DELETE FROM nome_tabella WHERE condizione;

Vediamo un esempio di utilizzo

```
<?php

$conn = mysqli_connect("localhost", "test", "test", "clienti");

$sql = "DELETE FROM anagrafica WHERE id = 1";

$result = mysqli_query($conn, $sql);

if(mysqli_affected_rows($conn) > 0) {

    echo "Utente eliminato con successo";
```

```
}

mysqli_close($conn);
```

Anche in questo caso tramite la funzione mysql_affected_rows() verifichiamo se ci sono state righe coinvolte nella cancellazione.

Se dopo aver cancellato il record eseguiamo una SELECT sulla tabella *anagrafica* otterremo un risultato simile al seguente

```
+----+------------+-----------+------+

| id | nome       | cognome   | eta  |

+----+------------+-----------+------+

|  2 | John       | Smith     |  32  |

|  3 | Pippo      | Bianchi   |  35  |

+----+------------+-----------+------+
```

dove l'utente con *id* uguale a 1 è stato cancellato.

> **Nota:** è importante specificare sempre la clausola WHERE in combinazione di DELETE, in quanto la sua omissione porterebbe all'eliminazione di tutti i record presenti della tabella.

Conclusioni

In questa guida abbiamo analizzato **l'estensione MySQLi**, ossia uno dei modi che PHP mette a disposizione per lavorare con un database MySQL. Abbiamo visto come connetterci ad una database ed effettuare semplici query per inserire, selezionare, modificare e cancellare i dati.

PDO, guida completa all'estensione PHP

Per accedere ad una base dati PHP offre due estensioni: MySQLi che abbiamo visto nella precedente guida e PDO che analizzeremo nel dettaglio in questa nuova guida.

Indice dei contenuti

1. Cos'è PDO
2. Database supportati
3. Vantaggi
4. Installazione
5. Connessione al database
6. Eseguire query
7. I metodi prepare(), execute() e fetchAll()
8. I prepared statement
9. Conclusioni

Cos'è PDO

PHP Data Objects (PDO) è un estensione PHP che fornisce un livello di astrazione di accesso ai dati, il che vuol dire che, indipendentemente dal database in uso, vengono utilizzate sempre le stesse funzioni per eseguire le query. PDO supporta più di 12 diversi DBMS attraverso l'utilizzo di uno driver specifico per quel database.

L'estensione è disponibile da PHP 5.1 e necessita di alcune caratteristiche tipiche della programmazione ad oggetti (OO) disponibili a partire dalla versione 5.0, quindi non funziona con versioni precedenti del linguaggio.

Database supportati

La seguente tabella elenca i DBMS supportati da PDO con relativo driver

Nome del driver	Database supportati
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC e win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 2 e SQLite 3
PDO_SQLSRV	Microsoft SQL Server / SQL Azure

Nome del driver	Database supportati
PDO_4D	4D

Anche se PDO offre la possibilità di cambiare DBMS senza dover riscrivere il codice, nella realtà non vi è quasi mai necessità di cambiare DBMS in corso d'opera. Tuttavia PDO presenta alcuni indiscutibili vantaggi che elenchiamo nel paragrafo successivo.

Vantaggi

PDO offre numerosi vantaggi che riguardano soprattuto la sicurezza delle applicazioni web e la fase di sviluppo e debug:

- **Sicurezza**: tramite i prepared statement è possibile prevenire la **SQL Injection**.
- **Riusabilità**: offre un'unica intefaccia per accedere a differenti database.
- **Flessibilità**: consente una gestione flessibile degli errori per la fase di debug.
- **Facilità di utilizzo**: rispetto a MySQLi sono necessarie solo due istruzioni per eseguire query in maniera sicura.
- **Usabilità**: offre numerosi helpers per compiere varie operazioni di routine.
- **Supporto** per i named parameters (:param) e i positional parameters (?) anche su database che non li supportano nativamente.

Nei paragrafi successivi vediamo nella pratica alcuni di questi vantaggi, iniziando dalla connessione al database e le classiche operazioni **CRUD** su di esso.

Installazione

Per poter utilizzare l'estensione PDO è necessario installare la versione specifica del database che si intende utilizzare. Per esempio nel caso di MySQL, dovrà essere installata l'estensione pdo_mysql e abilitarla dal file di configurazione php.ini.

Sarà sufficiente aprire con un editor il **file php.ini**, cercare la seguente riga

```
;extension=pdo_mysql
```

quindi rimuovere il punto e virgola iniziale, salvare il file e riavviare Apache per rendere effettive le modifiche.

Connessione al database

Per aprire la connessione al database è necessario creare un'instanza della classe PDO, nell'esempio che segue e in quelli successivi utilizzeremo il driver di MySQL per connetterci al database

```
<?php
try{

    $pdo = new PDO("mysql:host=localhost;dbname=shop", "test", "pass1234");

    // Set the PDO error mode to exception

    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
} catch(PDOException $e){

    die("ERRORE: Impossibile stabilire una connessione al database");

}
```

Come possiamo notare abbiamo introdotto due nuovi costrutti, uno per creare un nuovo oggetto tramite la keyword new e il costrutto try/catch per gestire le eccezioni, argomenti che vedremo nella guida avanzata.

In questa guida ci limitiamo a dire che il costrutto try/catch serve per interrompere l'esecuzione di un programma qualora l'istruzione contenuta dentro il blocco try dovesse generare un errore a runtime. In questo caso anziché mostrare all'utente un **errore fatale oppure un warning**, viene visualizzato un messaggio personalizzato e più leggibile che migliora l'usabilità dell'applicazione web.

Ritornando al nostro codice, alla riga 3 creiamo un'istanza della classe PDO rappresentante una connessione al database *shop* da parte dell'utente *test* con password *pass1234*, mentre *host* è uguale all'indirizzo del server MySQL. Nel nostro caso lavoriamo in locale, quindi il valore da inserire è *localhost.*

Come per le **funzioni**, anche per gli oggetti è possibile passare degli argomenti, in questo caso chiamati parametri.

Nel caso di PDO abbiamo:

PDO ($dsn, $username, $passwd, $options)

- $dsn: Data Source Name, rappresenta le informazioni che indicano ad un programma come connettersi ad una determinata fonte dati tramite un driver. In generale, un DSN è costituito dal nome del driver PDO, seguito da due punti e dalla sintassi di connessione specifica del driver PDO.
- $username: lo username dell'utente che deve connettersi alla base dati
- $passwd: la password dell'utente che deve connettersi alla base dati
- $options: Un'array associativo contentente alcune opzioni di connessione specifiche del driver.

Le opzioni possono essere impostate pure con il metodo setAttribute() della riga 6.

> **Nota:** l'impostazione dell'attributo PDO::ATTR_ERRMODE su PDO::ERRMODE_EXCEPTION dice a PDO di generare eccezioni ogni volta che si verifica un errore.

Un'altra opzione largamente usata è

PDO::MYSQL_ATTR_INIT_COMMAND

Ad esempio, per specificare l'utilizzo di caratteri UTF-8 è possibile scrivere il seguente codice

```
<?php

try{
```

```
    $pdo = new PDO(

        "mysql:host=localhost;dbname=shop",

        "test",

        "pass1234",

        [PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"] // La connessione utilizzerà la codifica UTF-8

    );

    // Set the PDO error mode to exception

    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

} catch(PDOException $e){

    die("Impossibile stabilire una connessione al database: " . $e.getMessage());

}
```

Una volta che abbiamo stabilito la connessione al database è possibile eseguire operazioni come inserimento, selezione, modifica e cancellazione.

Eseguire query

Eseguire query con PDO è molto semplice, è possibile utilizzare il metodo exec() per effettuare interrogazioni SQL di tipo INSERT, CREATE, UPDATE e DELETE, mentre il metodo query() consente di eseguire interrogazioni SELECT, che ci consentono di estrarre i dati dalla nostra base dati.

Diamo un'occhiata ad alcuni esempi di utilizzo. Per prima cosa creiamo la tabella *prodotti* nel nostro database *shop*

```
<?php

$sql = "CREATE TABLE `shop`.`prodotti` (

    `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
    `nome` VARCHAR(300) NOT NULL,

    `prezzo` DECIMAL(6,2) NOT NULL,

    PRIMARY KEY (`id`)

) ENGINE = InnoDB CHARSET=utf8mb4 COLLATE utf8mb4_general_ci;";

$result = $pdo->exec($sql);

if ($result) {

    echo "Tabella creata con successo";

}
```

Il codice precedente non fa altro che eseguire l'istruzione CREATE TABLE che consente la creazione di una nuova tabella MySQL.

In questo caso abbiamo usato il metodo exec() che prende come parametro un'istruzione SQL e restituisce il numero di righe interessate

```
<?php

$sql = 'INSERT INTO prodotti (nome, prezzo) VALUES ("sapone liquido 100ml", 8.50)';

$affectedRows = $connection->exec();

echo $affectedRows;
```

Il metodo query() viene usato per eseguire istruzioni SQL SELECT, vediamo un esempio

```
<?php

$sql = 'SELECT nome, prezzo FROM prodotti';

$products = $conn->query($sql);
```

```
foreach ($products as $product) {

    echo $product['nome'] . " " . $product['prezzo'] . "<br>";

}
```

Anche se può sembrare comodo, l'utilizzo di exec() e query() è sconsigliato per motivi di sicurezza, in quanto potenzialmente vulnerabili ad attacchi di tipo **SQL Injection**.

Introduciamo altri due metodi della classe PDO che consentono di eseguire query SQL in maniera sicura.

I metodi prepare() e execute() e fetchAll()

Per eseguire interrogazioni SQL in maniera sicura è consigliabile utilizzare in combinazione i metodi prepare() e execute() della classe PDO.

Il metodo prepare() prende in input un'istruzione SQL e restituisce un oggetto di tipo PDOStatement. Quest'oggetto può essere utilizzato per eseguire una query SQL.

Vediamo subito un esempio in combinazione con il metodo execute() per eseguire una SELECT dalla tabella *prodotti*

```
<?php

$sql = 'SELECT nome, prezzo FROM prodotti';

$sth = $conn->prepare($sql);

$result = $sth->execute();

if (! $result) {

    die('Errore esecuzione query: ' . implode(',', $conn->errorInfo()));

}

$products = $sth->fetchAll(PDO::FETCH_OBJ);

foreach ($products as $product) {
```

```
    echo $product->nome . " " . $product->prezzo . "<br>";


}
```

Una volta che è stata eseguita con successo la query attraverso l'istruzione execute(), che restituisce true in caso di successo, possiamo recuperare il risultato tramite il metodo fetchAll().

L'istruzione fetchAll() consente di recuperare tutti risultati della query SELECT, restituendo i valori sotto forma di oggetti della classe stdClass del linguaggio, la costante PDO::FETCH_OBJ passata come parametro alla funzione fetchAll() consente di fare questo.

Se invece vogliamo farci restituire i risultati in un'array associativo dove ciascun indice rappresenta il nome della colonna della tabella MySQL dobbiamo usare la costante PDO::FETCH_ASSOC al posto di PDO::FETCH_OBJ e per accedere al dato dobbiamo sostituire

$product['nome'] al posto di $product->nome

Il codice riportato in precedenza può essere usato per eseguire tutte le tipologie di interrogazioni SQL, dalla SELECT, alle INSERT, UPDATE e DELETE.

Vediamo ora alcuni esempi di codice in cui utilizziamo prepare() e execute() sfruttando i cosiddetti prepared statement.

I prepared statement

I prepared statement, letteralmente istruzioni preparate, consentono di "preparare" una query SQL utilizzando dei marcatori al posto dei dati. Tali marcatori sono rappresentati da un nome di variabile preceduto da due punti (named parameters), oppure da un punto interrogativo (positional parameters).

Vediamo meglio un esempio di utilizzo

```
<?php


$sql = 'SELECT nome, prezzo FROM prodotti WHERE id = :id';


$params = ['id' => 2];


$sth = $conn->prepare($sql);


$result = $sth->execute($params);


if (! $result) {
```

```
    die('Errore esecuzione query: ' . implode(',', $conn->errorInfo()));

}

$product = $sth->fetch();

if ($product) {

    echo $product['nome'] . " " . $product['prezzo'];

}
```

Nella query alla riga 2, abbiamo utilizzato il marcatore :id al posto del valore 2 che invece abbiamo inserito in un'array di parametri alla riga 3.

PDO non fa altro che sostituire il marcatore presente nella query con il relativo valore presente nell'array $params.

La sicurezza nell'utilizzo delle prepared statement sta nel fatto che in caso di valori provenienti da un input utente, tramite ad esempio la variabile $_GET, viene scongiurato l'inserimento di stringhe direttamente nella query da parte di malintenzionati che potrebbero hackerare la nostra applicazione.

Login e registrazione con Php e MySQLi (ultima versione) con file zippato della procedura

Oggi condividerò Simple User Registration & Login Script in PHP e MySQLi, https://www.allphptricks.com/simple-user-registration-login-script-in-php-and-mysqli/

spiegherò le funzionalità di base della registrazione e dell'accesso all'area protetta. In questo tutorial l'utente può registrarsi, accedere all'area riservata e disconnettersi.

Sto usando **PHP 5.6** per questo tutorial, non ho controllato questo tutorial su PHP 7, assicurati gentilmente di utilizzare **PHP 5.6** per evitare errori imprevisti.

Passaggi per creare una semplice registrazione utente e script di accesso in PHP e MySQLi

1. Crea una banca dati
2. Crea una tabella di database con i seguenti cinque campi:
   id, username, email, password, trn_date
3. Crea un modulo di registrazione
4. Crea un modulo di accesso
5. Connettiti al database utilizzando PHP
6. Autenticare l'utente connesso
7. Crea pagina indice
8. Crea pagina dashboard
9. Crea un logout
10. Crea un file CSS

1. Crea un database

Per creare un database, accedi a **phpmyadmin** e fai clic sulla scheda del database, inserisci il nome del tuo database e fai clic sul pulsante Crea database o semplicemente esegui la query seguente. Nel mio caso ho creato un database con il nome **register** .

```
CREATE DATABASE register;
```

2. Creazione di una tabella di database

Per creare una tabella, eseguire la query seguente in SQL.

```
CREATE TABLE IF NOT EXISTS `users` (
```

```
 `id` int(11) NOT NULL AUTO_INCREMENT,

 `username` varchar(50) NOT NULL,

 `email` varchar(50) NOT NULL,

 `password` varchar(50) NOT NULL,

 `trn_date` datetime NOT NULL,

 PRIMARY KEY (`id`)

 );
```

3. Creazione di un modulo di registrazione

Crea semplicemente una pagina con il nome **registration.php** e incolla il seguente codice al suo interno.

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Registration</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<?php

require('db.php');

// If form submitted, insert values into the database.

if (isset($_REQUEST['username'])){
```

```
        // removes backslashes

        $username = stripslashes($_REQUEST['username']);

        //escapes special characters in a string

        $username = mysqli_real_escape_string($con,$username);

        $email = stripslashes($_REQUEST['email']);

        $email = mysqli_real_escape_string($con,$email);

        $password = stripslashes($_REQUEST['password']);

        $password = mysqli_real_escape_string($con,$password);

        $trn_date = date("Y-m-d H:i:s");

        $query = "INSERT into `users` (username, password,
email, trn_date)

VALUES ('$username', '".md5($password)."', '$email',
'$trn_date')";

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

        if($result){

            echo "<div class='form'>

<h3>You are registered successfully.</h3>

<br/>Click here to <a href='login.php'>Login</a></div>";

        }

    }else{

?>

<div class="form">
```

```
<h1>Registration</h1>

<form name="registration" action="" method="post">

<input type="text" name="username" placeholder="Username" requir
ed />

<input type="email" name="email" placeholder="Email" required />

<input type="password" name="password" placeholder="Password" re
quired />

<input type="submit" name="submit" value="Register" />

</form>

</div>

<?php } ?>

</body>

</html>
```

IMG

4. Creazione di un modulo di accesso

Crea una pagina con nome **login.php** e incolla il seguente codice al suo interno.

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Login</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<?php

require('db.php');

session_start();

// If form submitted, insert values into the database.

if (isset($_POST['username'])){

        // removes backslashes

        $username = stripslashes($_REQUEST['username']);

        //escapes special characters in a string

        $username = mysqli_real_escape_string($con,$username);

        $password = stripslashes($_REQUEST['password']);

        $password = mysqli_real_escape_string($con,$password);

        //Checking is user existing in the database or not

        $query = "SELECT * FROM `users` WHERE
username='$username'
```

```
and password='".md5($password)."'";

        $result
= mysqli_query($con,$query) or die(mysql_error());

        $rows = mysqli_num_rows($result);

        if($rows==1){

            $_SESSION['username'] = $username;

            // Redirect user to index.php

            header("Location: index.php");

         }else{

        echo "<div class='form'>

<h3>Username/password is incorrect.</h3>

<br/>Click here to <a href='login.php'>Login</a></div>";

        }

    }else{

?>

<div class="form">

<h1>Log In</h1>

<form action="" method="post" name="login">

<input type="text" name="username" placeholder="Username" requir
ed />

<input type="password" name="password" placeholder="Password" re
quired />

<input name="submit" type="submit" value="Login" />
```

```
</form>

<p>Not registered yet? <a href='registration.php'>Register
Here</a></p>

</div>

<?php } ?>

</body>

</html>
```

IMG

5. Connettersi al database

Crea una pagina con il nome **db.php** e incolla il codice qui sotto.

```
<?php

// Enter your Host, username, password, database below.

// I left password empty because i do not set password on
localhost.

$con = mysqli_connect("localhost","root","","register");

// Check connection
```

```
if (mysqli_connect_errno())

  {

  echo "Failed to connect to MySQL: " . mysqli_connect_error();

  }

?>
```

6. Autenticare l'utente connesso

Crea una pagina con il nome **auth.php** e incolla il codice qui sotto.

```
<?php

session_start();

if(!isset($_SESSION["username"])){

header("Location: login.php");

exit(); }

?>
```

7. Creazione della pagina indice

Crea una pagina con il nome **index.php** e incolla il codice qui sotto.

```
<?php

//include auth.php file on all secure pages

include("auth.php");

?>

<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="utf-8">
<title>Welcome Home</title>
<link rel="stylesheet" href="css/style.css" />
</head>
<body>
<div class="form">
<p>Welcome <?php echo $_SESSION['username']; ?>!</p>
<p>This is secure area.</p>
<p><a href="dashboard.php">Dashboard</a></p>
<a href="logout.php">Logout</a>
</div>
</body>
</html>
```

IMG

8. Creazione della pagina del dashboard

Crea una pagina con il nome **dashboard.php** e incolla il codice sottostante.

```
<?php

require('db.php');

include("auth.php");

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Dashboard - Secured Page</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p>Dashboard</p>

<p>This is another secured page.</p>

<p><a href="index.php">Home</a></p>

<a href="logout.php">Logout</a>

</div>

</body>

</html>
```

IMG

9. Creazione di un logout

Crea una pagina con il nome **logout.php** e incolla il seguente codice al suo interno.

```
<?php

session_start();

// Destroying All Sessions

if(session_destroy())

{

// Redirecting To Home Page

header("Location: login.php");

}

?>
```

10. Creazione di un file CSS

Crea una pagina con il nome **style.css** e incolla il codice sottostante.

```
body {

     font-family:Arial, Sans-Serif;

}

.clearfix:before, .clearfix:after{
```

```
    content: "";
    display: table;
}
.clearfix:after{
    clear: both;
}
a{
    color:#0067ab;
    text-decoration:none;
}
a:hover{
    text-decoration:underline;
}
.form{
    width: 300px;
    margin: 0 auto;
}
input[type='text'], input[type='email'],
input[type='password'] {
    width: 200px;
    border-radius: 2px;
```

```
    border: 1px solid #CCC;

    padding: 10px;

    color: #333;

    font-size: 14px;

    margin-top: 10px;

}

input[type='submit']{

    padding: 10px 25px 8px;

    color: #fff;

    background-color: #0067ab;

    text-shadow: rgba(0,0,0,0.24) 0 1px 0;

    font-size: 16px;

    box-shadow: rgba(255,255,255,0.24) 0 2px 0 0 inset,#fff 0
1px 0 0;

    border: 1px solid #0164a5;

    border-radius: 2px;

    margin-top: 10px;

    cursor:pointer;

}

input[type='submit']:hover {

    background-color: #024978;

}
```

Nota: in questo tutorial il file CSS viene inserito nella cartella css/style.css. Quindi assicurati di aver inserito anche il file CSS nella stessa cartella css.

Clicca qui per scaricare il file compresso

Inserisci, visualizza, modifica ed elimina record dal database utilizzando PHP e MySQLi e file zippato

Link originale: https://www.allphptricks.com/insert-view-edit-and-delete-record-from-database-using-php-and-mysqli/

In questo tutorial, spiegherò come inserire, visualizzare, modificare ed eliminare record dal database utilizzando PHP e MySQLi, in pratica questo tutorial è una seconda parte di **Simple User Registration & Login Script in PHP e MySQLi** , in questa prima parte ho spiegato come per creare una semplice registrazione utente e accedere utilizzando PHP e MySQLi, se non sai come creare la registrazione utente e il modulo di accesso utente, quindi prima controlla questo tutorial Registrazione utente **semplice e script di accesso in PHP e MySQLi** , dopodiché ora torna a questo tutorial.

Sto usando **PHP 5.6** per questo tutorial, non ho controllato questo tutorial su PHP 7, assicurati gentilmente di utilizzare **PHP 5.6** per evitare errori imprevisti.

Ti suggerirei di controllare **PHP CRUD Operations Using PDO Prepared Statements** , poiché è compatibile con PHP 7 e 8 entrambi.

Passaggi per creare un record di inserimento, visualizzazione, modifica ed eliminazione dal database utilizzando PHP e MySQLi

Ora presumo che tu abbia già creato la registrazione dell'utente e i moduli di accesso che ho creato in **Simple User Registration & Login Script in PHP e MySQLi** ora creerò un'altra tabella per conservare i record, aggiornerò il file **dashboard.php** e aggiungerò altri quattro nuovi pagine che sono **insert.php, view.php, edit.php** e **delete.php** , procedi nel seguente modo:

1. Crea un'altra tabella per i record

2. Aggiorna file dashboard

3. Crea pagina di inserimento
4. Crea Visualizza pagina
5. Crea pagina Modifica/Aggiorna
6. Crea Elimina pagina

1. Creare un'altra tabella per i record

Esegui la seguente query SQL:

```
CREATE TABLE IF NOT EXISTS `new_record` (

 `id` int(11) NOT NULL AUTO_INCREMENT,

 `trn_date` datetime NOT NULL,

 `name` varchar(50) NOT NULL,

 `age`int(11) NOT NULL,

 `submittedby` varchar(50) NOT NULL,

 PRIMARY KEY (`id`)

 );
```

2. Aggiorna la pagina del dashboard

Aggiorna il file **dashboard.php** e incolla il seguente codice al suo interno.

```
<?php

require('db.php');

include("auth.php");

?>
```

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Dashboard - Secured Page</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p>Welcome to Dashboard.</p>

<p><a href="index.php">Home</a><p>

<p><a href="insert.php">Insert New Record</a></p>

<p><a href="view.php">View Records</a><p>

<p><a href="logout.php">Logout</a></p>

</div>

</body>

</html>
```

3. Creare una pagina di inserimento

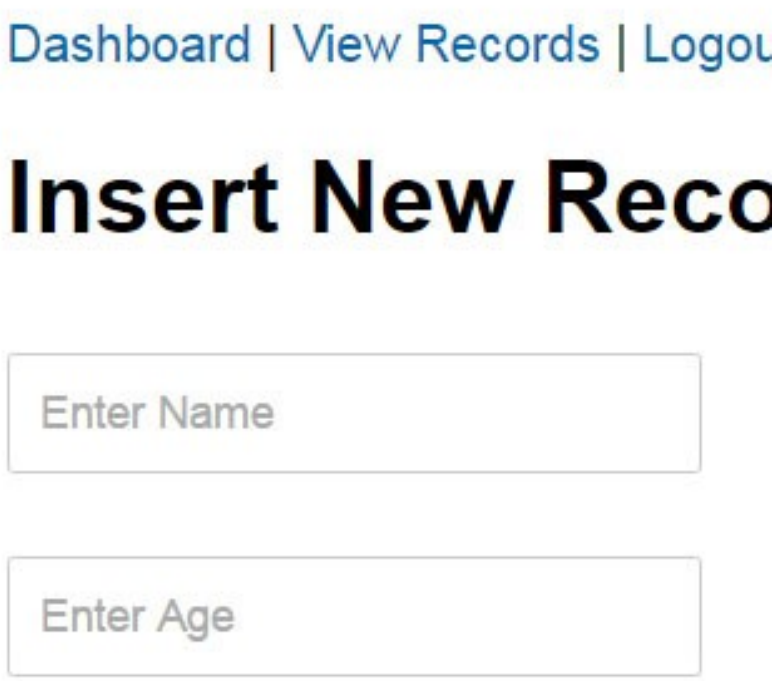
Crea una pagina con il nome **insert.php** e incolla il codice qui sotto.

```
<?php
```

```
require('db.php');

include("auth.php");

$status = "";

if(isset($_POST['new']) && $_POST['new']==1){

    $trn_date = date("Y-m-d H:i:s");

    $name =$_REQUEST['name'];

    $age = $_REQUEST['age'];

    $submittedby = $_SESSION["username"];

    $ins_query="insert into new_record

    (`trn_date`,`name`,`age`,`submittedby`)values

    ('$trn_date','$name','$age','$submittedby')";

    mysqli_query($con,$ins_query)

    or die(mysql_error());

    $status = "New Record Inserted Successfully.

    </br></br><a href='view.php'>View Inserted Record</a>";

}

?>

<!DOCTYPE html>

<html>

<head>
```

```
<meta charset="utf-8">

<title>Insert New Record</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="dashboard.php">Dashboard</a>

| <a href="view.php">View Records</a>

| <a href="logout.php">Logout</a></p>

<div>

<h1>Insert New Record</h1>

<form name="form" method="post" action="">

<input type="hidden" name="new" value="1" />

<p><input type="text" name="name" placeholder="Enter
Name" required /></p>

<p><input type="text" name="age" placeholder="Enter
Age" required /></p>

<p><input name="submit" type="submit" value="Submit" /></p>

</form>

<p style="color:#FF0000;"><?php echo $status; ?></p>

</div>
```

```
</div>

</body>

</html>
```



4. Crea Visualizza pagina

Crea una pagina con il nome **view.php** e incolla il codice qui sotto.

```
<?php

require('db.php');

include("auth.php");
```

```
?>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>View Records</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="index.php">Home</a>

| <a href="insert.php">Insert New Record</a>

| <a href="logout.php">Logout</a></p>

<h2>View Records</h2>

<table width="100%" border="1" style="border-
collapse:collapse;">

<thead>

<tr>

<th><strong>S.No</strong></th>

<th><strong>Name</strong></th>
```

```
<th><strong>Age</strong></th>

<th><strong>Edit</strong></th>

<th><strong>Delete</strong></th>

</tr>

</thead>

<tbody>

<?php

$count=1;

$sel_query="Select * from new_record ORDER BY id desc;";

$result = mysqli_query($con,$sel_query);

while($row = mysqli_fetch_assoc($result)) { ?>

<tr><td align="center"><?php echo $count; ?></td>

<td align="center"><?php echo $row["name"]; ?></td>

<td align="center"><?php echo $row["age"]; ?></td>

<td align="center">

<a href="edit.php?id=<?php echo $row["id"]; ?>">Edit</a>

</td>

<td align="center">

<a href="delete.php?id=<?php echo $row["id"]; ?>">Delete</a>

</td>
```

```
</tr>

<?php $count++; } ?>

</tbody>

</table>

</div>

</body>

</html>
```

Home | Insert New Record | Logout

View Records

S.No	Name	Age	Edit	D
1	Adnan	30	Edit	D
2	Imran	18	Edit	D
3	Kashif	25	Edit	D
4	Kamran	25	Edit	D

5. Crea pagina Modifica/Aggiorna

Crea una pagina con il nome **edit.php** e incolla il codice qui sotto.

```php
<?php

require('db.php');

include("auth.php");

$id=$_REQUEST['id'];

$query = "SELECT * from new_record where id='".$id."'";

$result = mysqli_query($con, $query) or die ( mysqli_error());

$row = mysqli_fetch_assoc($result);

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Update Record</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="dashboard.php">Dashboard</a>

| <a href="insert.php">Insert New Record</a>

| <a href="logout.php">Logout</a></p>
```

```
<h1>Update Record</h1>

<?php

$status = "";

if(isset($_POST['new']) && $_POST['new']==1)

{

$id=$_REQUEST['id'];

$trn_date = date("Y-m-d H:i:s");

$name =$_REQUEST['name'];

$age =$_REQUEST['age'];

$submittedby = $_SESSION["username"];

$update="update new_record set trn_date='".$trn_date."',

name='".$name."', age='".$age."',

submittedby='".$submittedby."' where id='".$id."'";

mysqli_query($con, $update) or die(mysqli_error());

$status = "Record Updated Successfully. </br></br>

<a href='view.php'>View Updated Record</a>";

echo '<p style="color:#FF0000;">'.$status.'</p>';

}else {

?>

<div>
```

```
<form name="form" method="post" action="">

<input type="hidden" name="new" value="1" />

<input name="id" type="hidden" value="<?php echo $row['id'];?>"
/>

<p><input type="text" name="name" placeholder="Enter Name"

required value="<?php echo $row['name'];?>" /></p>

<p><input type="text" name="age" placeholder="Enter Age"

required value="<?php echo $row['age'];?>" /></p>

<p><input name="submit" type="submit" value="Update" /></p>

</form>

<?php } ?>

</div>

</div>

</body>

</html>
```

Dashboard | Insert New Record | L

Update Record

Adnan

30

6. Crea Elimina pagina

Crea una pagina con il nome **delete.php** e incolla il codice qui sotto.

```
<?php

require('db.php');

$id=$_REQUEST['id'];

$query = "DELETE FROM new_record WHERE id=$id";

$result = mysqli_query($con,$query) or die ( mysqli_error());

header("Location: view.php");
```

```
?>
```

SCARICA DEMO

Link originale: https://www.allphptricks.com/insert-view-edit-and-delete-record-from-database-using-php-and-mysqli/

In questo tutorial, spiegherò come inserire, visualizzare, modificare ed eliminare record dal database utilizzando PHP e MySQLi, in pratica questo tutorial è una seconda parte di **Simple User Registration & Login Script in PHP e MySQLi** , in questa prima parte ho spiegato come per creare una semplice registrazione utente e accedere utilizzando PHP e MySQLi, se non sai come creare la registrazione utente e il modulo di accesso utente, quindi prima controlla questo tutorial Registrazione utente **semplice e script di accesso in PHP e MySQLi** , dopodiché ora torna a questo tutorial.

Sto usando **PHP 5.6** per questo tutorial, non ho controllato questo tutorial su PHP 7, assicurati gentilmente di utilizzare **PHP 5.6** per evitare errori imprevisti.

Ti suggerirei di controllare **PHP CRUD Operations Using PDO Prepared Statements** , poiché è compatibile con PHP 7 e 8 entrambi.

Passaggi per creare un record di inserimento, visualizzazione, modifica ed eliminazione dal database utilizzando PHP e MySQLi

Ora presumo che tu abbia già creato la registrazione dell'utente e i moduli di accesso che ho creato in **Simple User Registration & Login Script in PHP e MySQLi** ora creerò un'altra tabella per conservare i record, aggiornerò il file **dashboard.php** e aggiungerò altri quattro nuovi pagine che sono **insert.php, view.php, edit.php** e **delete.php** , procedi nel seguente modo:

1. Crea un'altra tabella per i record

2. Aggiorna file dashboard

3. Crea pagina di inserimento
4. Crea Visualizza pagina
5. Crea pagina Modifica/Aggiorna
6. Crea Elimina pagina

1. Creare un'altra tabella per i record

Esegui la seguente query SQL:

```
CREATE TABLE IF NOT EXISTS `new_record` (

 `id` int(11) NOT NULL AUTO_INCREMENT,

 `trn_date` datetime NOT NULL,

 `name` varchar(50) NOT NULL,

 `age`int(11) NOT NULL,

 `submittedby` varchar(50) NOT NULL,

 PRIMARY KEY (`id`)

 );
```

2. Aggiorna la pagina del dashboard

Aggiorna il file **dashboard.php** e incolla il seguente codice al suo interno.

```
<?php

require('db.php');

include("auth.php");

?>
```

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Dashboard - Secured Page</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p>Welcome to Dashboard.</p>

<p><a href="index.php">Home</a><p>

<p><a href="insert.php">Insert New Record</a></p>

<p><a href="view.php">View Records</a><p>

<p><a href="logout.php">Logout</a></p>

</div>

</body>

</html>
```

3. Creare una pagina di inserimento

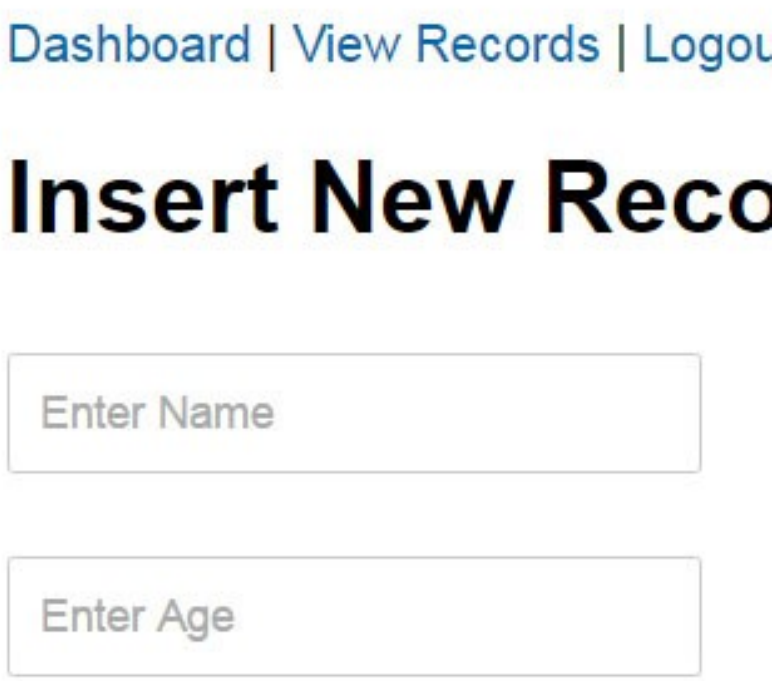
Crea una pagina con il nome **insert.php** e incolla il codice qui sotto.

```
<?php
```

```
require('db.php');

include("auth.php");

$status = "";

if(isset($_POST['new']) && $_POST['new']==1){

    $trn_date = date("Y-m-d H:i:s");

    $name =$_REQUEST['name'];

    $age = $_REQUEST['age'];

    $submittedby = $_SESSION["username"];

    $ins_query="insert into new_record

    (`trn_date`,`name`,`age`,`submittedby`)values

    ('$trn_date','$name','$age','$submittedby')";

    mysqli_query($con,$ins_query)

    or die(mysql_error());

    $status = "New Record Inserted Successfully.

    </br></br><a href='view.php'>View Inserted Record</a>";

}

?>

<!DOCTYPE html>

<html>

<head>
```

```
<meta charset="utf-8">

<title>Insert New Record</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="dashboard.php">Dashboard</a>

| <a href="view.php">View Records</a>

| <a href="logout.php">Logout</a></p>

<div>

<h1>Insert New Record</h1>

<form name="form" method="post" action="">

<input type="hidden" name="new" value="1" />

<p><input type="text" name="name" placeholder="Enter
Name" required /></p>

<p><input type="text" name="age" placeholder="Enter
Age" required /></p>

<p><input name="submit" type="submit" value="Submit" /></p>

</form>

<p style="color:#FF0000;"><?php echo $status; ?></p>

</div>
```

```
</div>

</body>

</html>
```



4. Crea Visualizza pagina

Crea una pagina con il nome **view.php** e incolla il codice qui sotto.

```
<?php

require('db.php');

include("auth.php");
```

```
?>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>View Records</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="index.php">Home</a>

| <a href="insert.php">Insert New Record</a>

| <a href="logout.php">Logout</a></p>

<h2>View Records</h2>

<table width="100%" border="1" style="border-
collapse:collapse;">

<thead>

<tr>

<th><strong>S.No</strong></th>

<th><strong>Name</strong></th>
```

```
<th><strong>Age</strong></th>

<th><strong>Edit</strong></th>

<th><strong>Delete</strong></th>

</tr>

</thead>

<tbody>

<?php

$count=1;

$sel_query="Select * from new_record ORDER BY id desc;";

$result = mysqli_query($con,$sel_query);

while($row = mysqli_fetch_assoc($result)) { ?>

<tr><td align="center"><?php echo $count; ?></td>

<td align="center"><?php echo $row["name"]; ?></td>

<td align="center"><?php echo $row["age"]; ?></td>

<td align="center">

<a href="edit.php?id=<?php echo $row["id"]; ?>">Edit</a>

</td>

<td align="center">

<a href="delete.php?id=<?php echo $row["id"]; ?>">Delete</a>

</td>
```

```
</tr>

<?php $count++; } ?>

</tbody>

</table>

</div>

</body>

</html>
```

Home | Insert New Record | Logout

View Records

S.No	Name	Age	Edit	D
1	Adnan	30	Edit	D
2	Imran	18	Edit	D
3	Kashif	25	Edit	D
4	Kamran	25	Edit	D

5. Crea pagina Modifica/Aggiorna

Crea una pagina con il nome **edit.php** e incolla il codice qui sotto.

```
<?php

require('db.php');

include("auth.php");

$id=$_REQUEST['id'];

$query = "SELECT * from new_record where id='".$id."'";

$result = mysqli_query($con, $query) or die ( mysqli_error());

$row = mysqli_fetch_assoc($result);

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Update Record</title>

<link rel="stylesheet" href="css/style.css" />

</head>

<body>

<div class="form">

<p><a href="dashboard.php">Dashboard</a>

| <a href="insert.php">Insert New Record</a>

| <a href="logout.php">Logout</a></p>
```

```
<h1>Update Record</h1>

<?php

$status = "";

if(isset($_POST['new']) && $_POST['new']==1)

{

$id=$_REQUEST['id'];

$trn_date = date("Y-m-d H:i:s");

$name =$_REQUEST['name'];

$age =$_REQUEST['age'];

$submittedby = $_SESSION["username"];

$update="update new_record set trn_date='".$trn_date."',

name='".$name."', age='".$age."',

submittedby='".$submittedby."' where id='".$id."'";

mysqli_query($con, $update) or die(mysqli_error());

$status = "Record Updated Successfully. </br></br>

<a href='view.php'>View Updated Record</a>";

echo '<p style="color:#FF0000;">'.$status.'</p>';

}else {

?>

<div>
```

```
<form name="form" method="post" action="">

<input type="hidden" name="new" value="1" />

<input name="id" type="hidden" value="<?php echo $row['id'];?>"
/>

<p><input type="text" name="name" placeholder="Enter Name"

required value="<?php echo $row['name'];?>" /></p>

<p><input type="text" name="age" placeholder="Enter Age"

required value="<?php echo $row['age'];?>" /></p>

<p><input name="submit" type="submit" value="Update" /></p>

</form>

<?php } ?>

</div>

</div>

</body>

</html>
```

Dashboard | Insert New Record | L

Update Record

Adnan

30

6. Crea Elimina pagina

Crea una pagina con il nome **delete.php** e incolla il codice qui sotto.

```
<?php

require('db.php');

$id=$_REQUEST['id'];

$query = "DELETE FROM new_record WHERE id=$id";

$result = mysqli_query($con,$query) or die ( mysqli_error());

header("Location: view.php");
```

?>

SCARICA DEMO

Conclusioni

Per chiudere questa guida possiamo dire che per l'accesso al database è preferibile utilizzare la classe PDO piuttosto che MySQLi, poiché garantisce una grande portabilità del codice grazie all'astrazione offerta dall'estensione stessa. Inoltre prepara chi è alle prime armi a iniziare a prendere confidenza con la programmazione ad oggetti, argomento di cui parleremo nella guida avanzata.

Con PDO abbiamo terminato la serie di guide base. Nelle prossime affronteremo argomenti più complessi tra cui le espressioni regolari, le eccezioni, la programmazione ad oggetti e tanto altro.