Stefano Braconi

```
<?php
    echo "<p>Semplicemente, PHP!</p>";
?>
```

Per chi vuole imparare

ma non sa da dove cominciare!

Edizione aggiornata al 29/06/2023

Sommario

Prefazione

Benvenuti nel mondo di PHP!

Questo manuale è stato creato pensando a voi, neofiti del linguaggio PHP. Se state leggendo queste parole, siete pronti per intraprendere un viaggio emozionante nel vasto universo della programmazione web.

PHP è un linguaggio di scripting potente e flessibile che vi consentirà di creare siti web dinamici e interattivi. Potrete manipolare i dati, interagire con database, gestire form e tanto altro ancora. Che abbiate una conoscenza di base della programmazione o siate completamente nuovi a questo mondo, questo manuale vi accompagnerà passo dopo passo, fornendovi le basi necessarie per padroneggiare PHP.

Durante il vostro percorso di apprendimento, ricordatevi che i fallimenti fanno parte del processo. Ogni errore è un'opportunità di crescita e di apprendimento. Non abbiate paura di sperimentare e mettere in pratica ciò che imparate. È attraverso la pratica e l'esplorazione che verrete a conoscenza delle potenzialità di PHP.

Desidero ringraziare il mio amico Fulvio, un uomo illuminato che illumina! La sua amicizia ha dato la scintilla per imparare la programmazione in PHP e mi ha spinto a condividere le mie conoscenze con voi.

Dedico il manuale a mia moglie Sonia e mia figlia Pantea Lyra.

Mi auguro che questo manuale vi ispiri e vi dia la fiducia necessaria per esplorare il mondo di PHP. Preparatevi ad aprire le porte a un universo di possibilità, la dove nessun programmatore è mai stato prima, dove le vostre idee prenderanno vita e le vostre creazioni prenderanno forma.

Buon viaggio nel mondo di PHP!

Stefano Braconi

1 - Introduzione a PHP

1. Che cos'è PHP

PHP (Hypertext Preprocessor) è un linguaggio di scripting lato server ampiamente utilizzato per lo sviluppo di applicazioni web dinamiche. È un linguaggio interpretato, il che significa che il codice PHP viene eseguito sul server web prima di inviare i risultati al browser dell'utente. PHP è stato progettato per essere integrato direttamente all'interno delle pagine HTML e può essere utilizzato per creare pagine web dinamiche, gestire i dati dei form, accedere a database e molto altro.

2. Vantaggi di utilizzare PHP

PHP offre diversi vantaggi per lo sviluppo di applicazioni web:

- Facilità di apprendimento: PHP ha una sintassi simile a molti altri linguaggi di programmazione, rendendolo relativamente facile da imparare per i principianti.

- Ampia compatibilità: PHP è compatibile con la maggior parte dei server web e dei sistemi operativi, rendendolo un'opzione flessibile per lo sviluppo web.

- Ampia comunità di sviluppatori: PHP ha una vasta comunità di sviluppatori che offrono supporto, condivisione di risorse e risposte a domande comuni.

- Open source: PHP è un linguaggio open source, il che significa che è gratuito da utilizzare e può essere personalizzato secondo le proprie esigenze.

- Integrabilità: PHP può essere facilmente integrato con diverse tecnologie, come database, framework e librerie esterne.

3. Installazione e configurazione di PHP

Per iniziare a utilizzare PHP, è necessario installare e configurare un server web sul proprio sistema. Alcune delle opzioni più comuni per l'installazione di PHP includono Apache, Nginx e IIS. Una volta installato il server web, sarà necessario configurare correttamente PHP per assicurarsi che venga eseguito correttamente. Questo coinvolge la modifica di file di configurazione specifici per indicare la posizione di PHP e altri settaggi.

4. Primi passi con PHP

Dopo aver installato e configurato correttamente PHP, è possibile creare un semplice file PHP con estensione ".php" per iniziare a scrivere codice. All'interno di questo file, è possibile includere codice PHP tra i tag “<?php” e “?>“. Il codice PHP può essere mescolato con l'HTML per generare output dinamico. Ad esempio:

```
<!DOCTYPE html>
<html>
<head>
    <title>Primo esempio PHP</title>
</head>
<body>
    <h1>Ciao, <?php echo "mondo!"; ?></h1>
</body>
</html>
```

5. Sintassi di base di PHP

PHP supporta una vasta gamma di funzionalità e costrutti, ma alcuni degli elementi di sintassi di base includono:

- Dichiarazione delle variabili: “$nome_variabile = valore;”

- Stampa di output: “echo "Testo da visualizzare";”

- Commenti: “// Commento su una singola linea” o “/* Commento su più linee */”

- Condizioni: “if (condizione) { // blocco di codice }”

- Cicli: “for (inizializzazione; condizione; incremento) { // blocco di codice }”

- Include di file: “include "nome_file.php";”

6. Variabili e tipi di dati

In PHP, le variabili sono utilizzate per memorizzare dati. Una variabile può contenere un valore di un determinato tipo di dati, come numeri, stringhe, booleani, array, oggetti e altro. Per assegnare un valore a una variabile, si utilizza l'operatore di assegnazione ("="). Ad esempio:

```
$nome = "Mario";
$eta = 25;
$pi_greco = 3.14159;
$verificato = true;
```

7. Operatori

PHP supporta una vasta gamma di operatori per eseguire operazioni su variabili e valori. Gli operatori includono gli operatori aritmetici (+, -, *, /), gli operatori di confronto (==, !=, <, >, <=, >=), gli operatori logici (&&, ||, !) e molti altri. Ad esempio:

```
$a = 10;
$b = 5;

// Operazioni aritmetiche
$sum = $a + $b; // 15
$subtraction = $a - $b; // 5
$multiplication = $a * $b; // 50
$division = $a / $b; // 2

// Operatori di confronto
$equal = ($a == $b); // false
$not_equal = ($a != $b); // true
$greater_than = ($a > $b); // true

// Operatori logici
$logical_and = ($a > 0 && $b > 0); // true
$logical_or = ($a > 0 || $b > 0); // true
$logical_not = !($a > 0); // false
```

8. Strutture di controllo

Le strutture di controllo consentono di prendere decisioni e controllare il flusso di esecuzione del programma. In PHP, le strutture di controllo più comuni includono le istruzioni condizionali “if”, “else if”, “else” e i cicli “for”, “while”, “do while”, “foreach”. Queste strutture consentono di eseguire blocchi di codice in base a determinate condizioni o di ripetere l'esecuzione di un blocco di codice per un certo numero di volte. Ad esempio:

```
$numero = 10;

if ($numero > 0) {
    echo "Il numero è positivo";
} elseif ($numero < 0) {
    echo "Il numero è negativo";
} else {
    echo "Il numero è zero";
}

for ($i = 0; $i < 5; $i++) {
    echo $i;
}

while ($condizione) {
    // blocco di codice
}

do {
    // blocco di codice
} while ($condizione);

foreach ($array as $elemento) {
    // blocco di codice
}
```

9. Funzioni

Le funzioni in PHP sono blocchi di codice che possono essere richiamati e riutilizzati in diversi punti del programma. Una funzione può avere parametri in ingresso e può restituire un valore di output. Definire le proprie funzioni permette di suddividere il codice in unità logiche e modulari, rendendo più facile la gestione e la manutenzione del codice.

Ecco un esempio di definizione di una funzione che calcola la somma di due numeri:

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

Questa funzione può essere chiamata passando due numeri come argomenti e restituirà il risultato della somma:

```
$risultato = somma(3, 4); // Il valore di $risultato sarà 7
```

Le funzioni possono anche avere parametri opzionali o predefiniti, consentendo di specificare valori di default nel caso in cui gli argomenti non vengano passati durante la chiamata della funzione. Inoltre, è possibile utilizzare la parola chiave “return” per restituire un valore dalla funzione.

10. Gestione degli errori e delle eccezioni

La gestione degli errori e delle eccezioni è un aspetto importante nella scrittura di codice PHP affidabile e robusto. PHP offre diversi meccanismi per gestire gli errori e le eccezioni al fine di identificare e gestire le situazioni anomale che possono verificarsi durante l'esecuzione del codice.

- Errori: Gli errori in PHP possono essere di diversi tipi, come errori di sintassi, errori di runtime o errori personalizzati. Per controllare come gli errori vengono gestiti nel codice, puoi utilizzare la funzione “error_reporting()” per impostare il livello di segnalazione degli errori e la funzione “set_error_handler()” per definire una funzione personalizzata che gestisce gli errori.

- Eccezioni: Le eccezioni consentono di gestire situazioni anomale o errori di runtime in modo più strutturato. Puoi utilizzare la struttura “try-catch” per catturare e gestire le eccezioni. Il blocco “try” contiene il codice che può generare un'eccezione, mentre il blocco “catch” viene eseguito se un'eccezione

viene generata. È possibile definire diverse sezioni “catch” per gestire eccezioni di tipi diversi. Inoltre, puoi utilizzare la parola chiave “throw” per generare manualmente un'eccezione in un punto specifico del codice.

Ecco un esempio di utilizzo delle eccezioni:

```
try {
    // Codice che potrebbe generare un'eccezione
    if ($condizione) {
        throw new Exception("Errore personalizzato");
    }
} catch (Exception $e) {
    // Gestione dell'eccezione
    echo "Si è verificato un errore: " . $e->getMessage();
}
```

In questo esempio, se la condizione specificata è vera, verrà generata un'eccezione personalizzata di tipo “Exception”, che verrà quindi catturata dal blocco “catch” per la gestione dell'errore.

È possibile personalizzare ulteriormente la gestione degli errori e delle eccezioni in base alle esigenze specifiche dell'applicazione, ad esempio registrando gli errori in un file di log o visualizzando messaggi di errore più dettagliati per il debugging.

2 - Programmazione orientata agli oggetti

1. Concetti fondamentali della programmazione orientata agli oggetti (POO)

La programmazione orientata agli oggetti è un paradigma di programmazione che si basa sulla creazione di oggetti che rappresentano entità del mondo reale o concetti astratti. Alcuni concetti fondamentali della POO includono:

- Incapsulamento: Consiste nel raggruppare dati (proprietà) e operazioni (metodi) all'interno di un oggetto, impedendo l'accesso diretto ai dati e consentendo di definire regole per manipolarli.

- Ereditarietà: Consente di creare nuove classi (classi derivate o sottoclassi) che ereditano le proprietà e i metodi da una classe esistente (classe base o superclasse), permettendo la condivisione del codice e l'estensione delle funzionalità.

- Polimorfismo: Consiste nella capacità di un oggetto di assumere diverse forme, cioè di rispondere in modi diversi a chiamate di metodi con lo stesso nome, a seconda del tipo effettivo dell'oggetto.

- Astrazione: Consiste nel fornire una visione semplificata di un oggetto complesso, focalizzandosi solo sulle caratteristiche e sui comportamenti rilevanti per l'applicazione, nascondendo i dettagli di implementazione.

2. Classi e oggetti

In POO, una classe è un modello o un prototipo che definisce le proprietà e i metodi comuni a un insieme di oggetti. Un oggetto è un'istanza di una classe, una vera entità che esiste in memoria. Ad esempio, se immaginiamo una classe “Auto”, un oggetto può essere un'auto specifica come una Fiat Panda. Una classe può avere più istanze di oggetti.

```
class Auto {
    // Proprietà
    public $marca;
    public $modello;

    // Metodo
    public function accendi() {
        echo "L'auto è stata accesa.";
    }
}

// Creazione di un oggetto Auto
$FiatPanda = new Auto();

// Accesso alle proprietà e chiamata del metodo
$FiatPanda->marca = "Fiat";
$FiatPanda->modello = "Panda";
$FiatPanda->accendi(); // Stampa "L'auto è stata accesa."
```

3. Proprietà e metodi

Le proprietà rappresentano i dati associati a un oggetto, mentre i metodi sono le funzioni o le azioni che un oggetto può compiere. Le proprietà sono definite all'interno della classe e possono essere pubbliche, private o protette, a seconda dell'accessibilità desiderata. I metodi, invece, definiscono il comportamento dell'oggetto e possono accedere e manipolare le proprietà. Ad esempio, una classe “Auto” potrebbe avere proprietà come “marca”, “modello” e metodi come “accendi()”, “spegni()”, “accelera()”.

```
class Auto {
    public $marca;
    private $modello;

    public function __construct($marca, $modello) {
        $this->marca = $marca;
        $this->modello = $modello;
    }

    public function getModello() {
        return $this->modello;
    }

    public function accelerazione() {
        echo "L'auto sta accelerando.";
    }
}

$FiatPanda = new Auto("Fiat", "Panda");
echo $FiatPanda->marca; // Stampa "Fiat"
echo $FiatPanda->getModello(); // Stampa "Panda"
$FiatPanda->accelerazione(); // Stampa "L'auto sta accelerando."
```

4. Ereditarietà e classi derivate

L'ereditarietà consente di creare nuove classi derivate (o sottoclassi) che ereditano le proprietà e i metodi da una classe base (o superclasse). Le classi derivate possono estendere o sovrascrivere il comportamento della classe base e aggiungere nuove proprietà o metodi. Questo favorisce il riutilizzo del codice e la creazione di gerarchie di classi.

```
class Veicolo {
    public $marca;

    public function __construct($marca) {
        $this->marca = $marca;
    }

    public function avvia() {
        echo "Il veicolo è stato avviato.";
    }
}

class Auto extends Veicolo {
    public function accelerazione() {
        echo "L'auto sta accelerando.";
    }
}

$FiatPanda = new Auto("Fiat");
$FiatPanda->avvia(); // Stampa "Il veicolo è stato avviato."
$FiatPanda->accelerazione(); // Stampa "L'auto sta accelerando."
```

5. Interfacce e classi astratte

Le interfacce definiscono un contratto che una classe deve implementare, specificando i metodi che la classe deve contenere. Una classe può implementare più interfacce, garantendo la conformità a diversi contratti. Le classi astratte, invece, sono classi parzialmente implementate che non possono essere istanziate direttamente, ma devono essere estese da altre classi.

```
interface Guidabile {
    public function accelerazione();
    public function frenata();
}

abstract class Veicolo implements Guidabile {
    public $marca;

    public function __construct($marca) {
        $this->marca = $marca;
    }

    abstract public function avvia();

    public function frenata() {
        echo "Il veicolo sta frenando.";
    }
}

class Auto extends Veicolo {
    public function avvia() {
        echo "L'auto è stata avviata.";
    }

    public function accelerazione() {
        echo "L'auto sta accelerando.";
    }
}

$FiatPanda = new Auto("Fiat");
$FiatPanda->avvia(); // Stampa "L'auto è stata avviata."
$FiatPanda->accelerazione(); // Stampa "L'auto sta accelerando."
$FiatPanda->frenata(); // Stampa "Il veicolo sta frenando."
```

6. Namespace

I namespace consentono di organizzare le classi, le funzioni e le costanti in gruppi separati per evitare collisioni di nomi. I namespace forniscono uno spazio dei nomi univoco per evitare conflitti di nomenclatura e migliorare la gestione dei nomi delle entità. Ad esempio, è possibile definire un namespace per un modulo specifico dell'applicazione e utilizzare il namespace per accedere alle classi all'interno di quel modulo.

```
namespace MioNamespace;

class MiaClasse {
    // ...
}

$oggetto = new MiaClasse();
```

7. Autoloaders

Gli autoloaders consentono di caricare automaticamente le classi quando vengono utilizzate per la prima volta nel codice. Possiamo utilizzare una funzione di autoloading personalizzata per gestire il caricamento automatico delle classi. Ad esempio, possiamo utilizzare la funzione “spl_autoload_register()” per registrare un autoloader che cerca automaticamente le classi all'interno di determinate directory.

```
spl_autoload_register(function ($nomeClasse) {
    $file = __DIR__ . '/path/to/classes/' . $nomeClasse . '.php';
    if (file_exists($file)) {
        include $file;
    }
});
```

Con l'utilizzo di autoloaders, non è necessario includere manualmente tutti i file delle classi, poiché le classi saranno caricate automaticamente al momento dell'utilizzo.

3 - Gestione delle stringhe e degli array

1. Operazioni sulle stringhe

Le stringhe sono sequenze di caratteri e in PHP è possibile eseguire numerose operazioni su di esse. Alcune operazioni comuni sulle stringhe includono:

- Concatenazione di stringhe: Per unire due stringhe si utilizza l'operatore di concatenazione (“.”). Ad esempio:

```
$nome = "Mario";
$cognome = "Rossi";
$nomeCompleto = $nome . " " . $cognome;
echo $nomeCompleto; // Stampa "Mario Rossi"
```

- Lunghezza di una stringa: Per determinare la lunghezza di una stringa si utilizza la funzione “strlen()”. Ad esempio:

```
$testo = "Ciao mondo!";
$lunghezza = strlen($testo);
echo $lunghezza; // Stampa "11"
```

- Accesso ai singoli caratteri: È possibile accedere ai singoli caratteri di una stringa utilizzando la sintassi “$stringa[indice]”. Gli indici iniziano da 0. Ad esempio:

```
$testo = "Ciao";
$primoCarattere = $testo[0];
echo $primoCarattere; // Stampa "C"
```

2. Manipolazione delle stringhe

PHP fornisce numerose funzioni per manipolare le stringhe e modificarne il contenuto. Alcune operazioni comuni includono:

- Conversione di maiuscole/minuscole: È possibile convertire una stringa in maiuscolo o minuscolo utilizzando le funzioni “strtoupper()” e “strtolower()”. Ad esempio:

```
$testo = "Ciao Mondo!";
$maiuscolo = strtoupper($testo);
$minuscolo = strtolower($testo);
echo $maiuscolo; // Stampa "CIAO MONDO!"
echo $minuscolo; // Stampa "ciao mondo!"
```

- Estrazione di sottostringhe: È possibile estrarre una parte di una stringa utilizzando la funzione “substr()”. Ad esempio:

```
$testo = "Ciao Mondo!";
$sottostringa = substr($testo, 5, 5);
echo $sottostringa; // Stampa "Mondo"
```

- Sostituzione di una sottostringa: È possibile sostituire una sottostringa con un'altra utilizzando la funzione “str_replace()”. Ad esempio:

```
$testo = "Ciao Mondo!";
$testoModificato = str_replace("Mondo", "Universo", $testo);
echo $testoModificato; // Stampa "Ciao Universo!"
```

3. Funzioni utili per la gestione delle stringhe

PHP offre una vasta gamma di funzioni per la gestione delle stringhe. Alcune funzioni utili includono:

- “strpos()”: Restituisce la posizione della prima occorrenza di una sottostringa all'interno di una stringa. Se la sottostringa non viene trovata, restituisce “false”.

- “strrev()”: Inverte una stringa.

- “trim()”: Rimuove gli spazi bianchi all'inizio e alla fine di una stringa.

- “str_pad()”: Aggiunge un padding a sinistra o a destra di una stringa.

- “str_split()”: Divide una stringa in un array di caratteri.

```
$frase = "Benvenuti a OpenAI!";
$posizione = strpos($frase, "OpenAI");
echo $posizione; // Stampa "13"

$fraseInvertita = strrev($frase);
echo $fraseInvertita; // Stampa "!IAnepO a itunevneB"

$testo = "   Ciao   ";
$testoTrimmed = trim($testo);
echo $testoTrimmed; // Stampa "Ciao"

$testoPadded = str_pad($testo, 10, "*");
echo $testoPadded; // Stampa "   Ciao***"

$arrayCaratteri = str_split($testo);
print_r($arrayCaratteri); // Stampa Array(" ", " ", "C", "i", "a", "o", " ", " ", " ")
```

4. Operazioni sugli array

Gli array sono strutture dati che permettono di gestire collezioni di elementi in PHP. Alcune operazioni comuni sugli array includono:

- Creazione di un array vuoto: È possibile creare un array vuoto utilizzando la sintassi "$nomeArray = array();" o la sintassi abbreviata "[]". Ad esempio:

```
$arrayVuoto = array();
// oppure
$arrayVuoto = [];
```

- Aggiunta di elementi all'array vuoto: Per aggiungere elementi all'array vuoto, è possibile utilizzare la funzione "array_push()". Ad esempio:

```
$arrayVuoto = array();
array_push($arrayVuoto, "elemento1", "elemento2");
print_r($arrayVuoto);
// Stampa Array("elemento1", "elemento2")
```

- Accesso agli elementi: È possibile accedere a un elemento di un array utilizzando la sintassi "$array[indice]". Gli indici iniziano da 0. Ad esempio:

```
$frutta = array("Mela", "Banana", "Arancia");
echo $frutta[0]; // Stampa "Mela"
```

- Altre operazioni sugli array: Le operazioni sugli array includono la rimozione di elementi ("unset()"), la verifica dell'esistenza di un elemento ("in_array()"), la fusione di array ("array_merge()") e molto altro.

4 - Gestione dei file e delle directory

1. Lettura e scrittura di file

PHP fornisce diverse funzioni per la lettura e la scrittura di file. Alcune operazioni comuni includono:

- Apertura di un file: Per aprire un file, si utilizza la funzione “fopen()”, che restituisce un puntatore al file. È possibile specificare la modalità di apertura del file, come "r" per la lettura o "w" per la scrittura. Ad esempio:

```
$handle = fopen("file.txt", "r");
```

- Lettura del contenuto di un file: È possibile leggere il contenuto di un file utilizzando la funzione “fread()”. È necessario specificare il puntatore al file e la quantità di dati da leggere. Ad esempio:

```
$handle = fopen("file.txt", "r");
$contenuto = fread($handle, filesize("file.txt"));
echo $contenuto;
```

- Scrittura in un file: È possibile scrivere nel file utilizzando la funzione “fwrite()”. È necessario specificare il puntatore al file e i dati da scrivere. Ad esempio:

```
$handle = fopen("file.txt", "w");
fwrite($handle, "Questo è un esempio di scrittura su file.");
fclose($handle);
```

2. Manipolazione dei file

PHP offre diverse funzioni per manipolare i file, tra cui:

- Rinominare un file: È possibile rinominare un file utilizzando la funzione “rename()”. È necessario specificare il nome originale del file e il nuovo nome. Ad esempio:

```
rename("file.txt", "nuovo_file.txt");
```

- Copiare un file: È possibile copiare un file in un'altra posizione utilizzando la funzione “copy()”. È necessario specificare il nome originale del file e il percorso di destinazione. Ad esempio:

```
copy("file.txt", "cartella/nuovo_file.txt");
```

- Eliminare un file: È possibile eliminare un file utilizzando la funzione “unlink()”. È necessario specificare il nome del file da eliminare. Ad esempio:

```
unlink("file.txt");
```

3. Gestione delle directory

PHP fornisce funzioni per la gestione delle directory, inclusa la creazione, l'eliminazione e l'elenco dei file contenuti. Alcune operazioni comuni includono:

- Creazione di una directory: È possibile creare una nuova directory utilizzando la funzione “mkdir()”. È necessario specificare il nome della directory e, se necessario, i permessi. Ad esempio:

```
mkdir("nuova_directory");
```

- Eliminare una directory: È possibile eliminare una directory vuota utilizzando la funzione “rmdir()”. È necessario specificare il nome della directory da eliminare. Ad esempio:

```
rmdir("directory_da_eliminare");
```

- Elenco dei file in una directory: È possibile ottenere un elenco dei file presenti in una directory utilizzando la funzione “scandir()”. Restituisce un array contenente i nomi dei file. Ad esempio:

```
$files = scandir("directory");
print_r($files);
```

4. Funzioni utili per la gestione dei file e delle directory

PHP offre numerose funzioni utili per la gestione dei file e delle directory. Alcune funzioni comuni includono:

- “file_exists()”: Verifica se un file o una directory esiste.

- “is_file()”: Verifica se un percorso corrisponde a un file.

- “is_dir()”: Verifica se un percorso corrisponde a una directory.

- “file_get_contents()”: Restituisce il contenuto di un file come stringa.

- “file_put_contents()”: Scrive una stringa in un file.

- “glob()”: Restituisce un elenco di file corrispondenti a un determinato pattern.

.

5 - Gestione dei dati con database

1. Connessione a un database

Per gestire i dati con database in PHP, è necessario stabilire una connessione al database desiderato. Di seguito viene illustrato come connettersi a un database utilizzando MySQL come esempio:

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "nome_database";

// Creazione della connessione
$conn = new mysqli($servername, $username, $password, $dbname);

// Verifica della connessione
if ($conn->connect_error) {
    die("Connessione fallita: " . $conn->connect_error);
}
```

2. Esecuzione di query SQL

Dopo aver stabilito la connessione al database, è possibile eseguire query SQL per manipolare i dati. Ecco alcuni esempi comuni:

- Esempio di SELECT:

```
$sql = "SELECT * FROM tabella";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "Nome: " . $row["nome"] . ", Cognome: " . $row["cognome"];
    }
} else {
    echo "Nessun risultato trovato.";
}
```

- Esempio di INSERT:

```
$sql = "INSERT INTO tabella (nome, cognome) VALUES ('Mario', 'Rossi')";
if ($conn->query($sql) === TRUE) {
    echo "Dati inseriti con successo.";
} else {
    echo "Errore durante l'inserimento dei dati: " . $conn->error;
}
```

- Esempio di UPDATE:

```
$sql = "UPDATE tabella SET nome='Nuovo Nome' WHERE id=1";
if ($conn->query($sql) === TRUE) {
    echo "Dati aggiornati con successo.";
} else {
    echo "Errore durante l'aggiornamento dei dati: " . $conn->error;
}
```

- Esempio di DELETE:

```
$sql = "DELETE FROM tabella WHERE id=1";
if ($conn->query($sql) === TRUE) {
    echo "Dati eliminati con successo.";
} else {
    echo "Errore durante l'eliminazione dei dati: " . $conn->error;
}
```

3. Prevenzione delle vulnerabilità di sicurezza

Quando si gestiscono dati con database, è importante prevenire le vulnerabilità di sicurezza, come gli attacchi di SQL injection. Per evitare questi problemi, è possibile utilizzare le query parametriche o i prepared statements. Di seguito viene presentato un esempio di prepared statement:

```
$stmt = $conn->prepare("SELECT * FROM tabella WHERE id = ?");
$stmt->bind_param("i", $id);

$id = 1;
$stmt->execute();
$result = $stmt->get_result();

while ($row = $result->fetch_assoc()) {
    echo "Nome: " . $row["nome"] . ", Cognome: " . $row["cognome"];
}

$stmt->close();
```

4. Funzioni utili per la gestione dei database

PHP fornisce una serie di funzioni utili per la gestione dei database. Alcune di queste includono:

- “mysqli_real_escape_string()”: Utilizzata per prevenire l'iniezione di SQL

e proteggere i dati da caratteri speciali.

- “mysqli_num_rows()”: Restituisce il numero di righe restituite da una query SELECT.

- “mysqli_affected_rows()”: Restituisce il numero di righe interessate da un'operazione di inserimento, aggiornamento o eliminazione.

- “mysqli_fetch_assoc()”: Restituisce una riga di risultato come un array associativo.

6 - Prevenzione delle SQL Injection

1. Cos'è la SQL Injection

La SQL Injection è una tecnica di attacco in cui un aggressore inserisce codice SQL dannoso all'interno di un'istruzione SQL. Questo avviene sfruttando l'input dell'utente che non viene adeguatamente validato o sanificato. L'obiettivo dell'attaccante è manipolare l'istruzione SQL in modo da ottenere accesso non autorizzato ai dati o eseguire operazioni dannose sul database.

2. Utilizzo dei prepared statements

I prepared statements sono un meccanismo sicuro per eseguire query SQL parametriche in modo da prevenire la SQL Injection. Invece di concatenare direttamente i valori degli input nell'istruzione SQL, si utilizzano segnaposto o parametri nella query. Il database quindi prepara l'istruzione SQL e associa i valori dei parametri separatamente.

Ecco un esempio di utilizzo dei prepared statements:

```
$stmt = $conn->prepare("SELECT * FROM utenti WHERE username = ? AND password = ?");
$stmt->bind_param("ss", $username, $password);

$username = $_POST['username'];
$password = $_POST['password'];

$stmt->execute();
$result = $stmt->get_result();
```

3. Uso dei parametri nella query

L'uso dei parametri nella query è fondamentale per prevenire la SQL Injection. I parametri sono segnaposto nella query che verranno sostituiti con i valori corrispondenti. È importante utilizzare il corretto tipo di parametro (stringa, intero, etc.) e associare i valori corretti prima di eseguire la query.

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

$nome = $_POST['nome'];
$cognome = $_POST['cognome'];

$stmt->execute();
$stmt->close();
```

4. Sanitizzazione dei dati di input

Oltre all'uso dei prepared statements, è buona pratica sanitizzare i dati di input per rimuovere eventuali caratteri dannosi. In PHP, è possibile utilizzare la funzione “mysqli_real_escape_string()” per effettuare l'escape dei caratteri speciali. Tuttavia, l'uso dei prepared statements è preferibile per la sicurezza dei dati.

```
$username = mysqli_real_escape_string($conn, $_POST['username']);
$password = mysqli_real_escape_string($conn, $_POST['password']);
```

5. Utilizzo di funzioni specifiche per la manipolazione dei dati

Inoltre, è importante utilizzare funzioni specifiche per la manipolazione dei dati quando necessario. Ad esempio, per gestire le password, è consigliabile utilizzare funzioni di hash come “password_hash()” per memorizzare le password in modo sicuro e “password_verify()” per confrontare le password immesse dagli utenti con quelle memorizzate nel database.

```
$hashedPassword = password_hash($password, PASSWORD_DEFAULT);
// Memorizzare $hashedPassword nel database
// Verificare la password immessa dall'utente
if (password_verify($password, $hashedPassword)) {
    // Password corretta
} else {
    // Password errata
}
```

10 - Librerie e framework PHP

1. Introduzione alle librerie e ai framework PHP

Le librerie e i framework PHP sono strumenti che offrono funzionalità predefinite e soluzioni comuni per semplificare lo sviluppo di applicazioni web. Sono progettati per consentire una maggiore produttività, modularità del codice e riutilizzo delle risorse. Le librerie sono raccolte di funzioni e classi che forniscono un insieme specifico di funzionalità, mentre i framework sono strutture complete che includono librerie, convenzioni e pattern di sviluppo.

2. Esempi di librerie e framework popolari

- Laravel: Un framework PHP completo e potente che offre un'ampia gamma di funzionalità per lo sviluppo di applicazioni web.

- Symfony: Un framework PHP modulare e ad alte prestazioni che favorisce il riutilizzo del codice e l'adozione delle migliori pratiche di sviluppo.

- CodeIgniter: Un framework PHP leggero e semplice da utilizzare, ideale per lo sviluppo rapido di applicazioni web.

- jQuery: Una libreria JavaScript ampiamente utilizzata per semplificare la manipolazione del DOM, la gestione degli eventi e le chiamate AJAX.

- Bootstrap: Una libreria CSS e JavaScript per la creazione di interfacce web moderne e responsive.

- Guzzle: Una libreria PHP per l'invio di richieste HTTP e l'interazione con API esterne.

3. Utilizzo di librerie e framework esterni

Per utilizzare librerie e framework esterni in PHP, di solito è necessario seguire questi passaggi:

- Scarica e installa la libreria o il framework desiderato. Questo può essere fatto manualmente o utilizzando un gestore di pacchetti come Composer.

- Importa le classi o le funzioni necessarie nel tuo codice utilizzando l'istruzione “require” o “use”, a seconda delle specifiche della libreria o del framework.

- Utilizza le funzionalità offerte dalla libreria o dal framework seguendo la documentazione fornita. Di solito, dovrai chiamare le funzioni, estendere le classi o configurare le impostazioni in base alle tue esigenze.

Ecco un esempio di utilizzo di una libreria esterna come Guzzle per eseguire una richiesta HTTP:

```
// Richiedi l'autoloader di Composer
require 'vendor/autoload.php';

use GuzzleHttp\Client;

// Crea un'istanza del client Guzzle
$client = new Client();

// Esegui una richiesta GET a un URL specifico
$response = $client->get('https://api.example.com/data');

// Ottieni il corpo della risposta
$body = $response->getBody()->getContents();

// Gestisci i dati della risposta
echo $body;
```

11 - Debugging e profiling

1. Debugging del codice PHP

Il debugging del codice PHP è un processo fondamentale per individuare e correggere gli errori o i bug presenti nell'applicazione. Ecco alcuni strumenti e tecniche comuni per il debugging:

- Utilizzo di “echo” o “var_dump”: Puoi inserire istruzioni di output nel tuo codice per visualizzare i valori delle variabili e le informazioni di debug durante l'esecuzione del programma:

```
$variable = "Hello, World!";
echo $variable;
```

- Log di debug: Puoi registrare i messaggi di debug in un file di log per analizzarli in seguito e individuare eventuali errori o problemi nel flusso di esecuzione del codice.

```
error_log("This is a debug message");
```

- Utilizzo di un debugger: Puoi utilizzare un debugger specifico per PHP, come Xdebug, per eseguire il codice passo-passo, impostare punti di interruzione e analizzare lo stato delle variabili durante l'esecuzione del programma.

2. Strumenti di profiling e ottimizzazione

Il profiling del codice è una tecnica che consente di analizzare le prestazioni dell'applicazione e individuare eventuali aree di miglioramento o di ottimizzazione. Di seguito sono riportati alcuni strumenti e tecniche comuni per il profiling:

- Strumenti di profiling: Esistono diversi strumenti di profiling disponibili per PHP, come Xdebug e Blackfire, che consentono di raccogliere dati sulle prestazioni dell'applicazione, inclusi tempi di esecuzione, utilizzo della memoria e chiamate di funzioni. Questi strumenti generano report dettagliati che possono essere utilizzati per identificare le parti del codice che richiedono miglioramenti.

- Ottimizzazione del codice: Una volta identificate le aree del codice che richiedono ottimizzazione, puoi apportare modifiche al tuo codice per migliorarne le prestazioni. Ciò può includere l'utilizzo di algoritmi più efficienti, la riduzione del numero di query al database, la memorizzazione nella cache dei risultati o l'eliminazione di codice ridondante o non utilizzato.

- Analisi delle query: Se l'applicazione interagisce con un database, è importante analizzare le query eseguite per individuare eventuali ottimizzazioni. Puoi utilizzare strumenti come EXPLAIN o Query Profiler per identificare le query lente, gli indici mancanti o le operazioni di join inefficienti.

- Monitoraggio delle prestazioni: Puoi utilizzare strumenti di monitoraggio delle prestazioni come New Relic o Datadog per ottenere una visione più completa delle prestazioni dell'applicazione. Questi strumenti forniscono dati in tempo reale sulle prestazioni, tra cui tempi di risposta, utilizzo delle risorse e statistiche di traffico.

È importante ricordare che il debugging e il profiling sono processi iterativi e continui. È consigliabile eseguire regolarmente il debugging del codice e il profiling delle prestazioni per garantire che l'applicazione funzioni correttamente e sia ottimizzata.

12 - Pubblicazione e deployment

1. Opzioni di deployment per applicazioni PHP

Quando si tratta di pubblicare e distribuire un'applicazione PHP, ci sono diverse opzioni disponibili. Alcune delle più comuni sono:

- Hosting condiviso: È possibile utilizzare un servizio di hosting condiviso per caricare l'applicazione su un server condiviso. Questa opzione è spesso economica ma può avere limitazioni in termini di risorse e flessibilità.

- Server dedicato: È possibile noleggiare un server dedicato e configurarlo per ospitare l'applicazione PHP. Questa opzione offre maggiore controllo e flessibilità rispetto all'hosting condiviso, ma richiede competenze tecniche per la gestione del server.

- Cloud hosting: I servizi di cloud hosting, come Amazon Web Services (AWS) o Google Cloud Platform (GCP), offrono la possibilità di distribuire l'applicazione su server virtuali nel cloud. Questa opzione offre scalabilità, ridondanza e flessibilità, consentendo di adattare le risorse alle esigenze dell'applicazione.

- Platform as a Service (PaaS): Le piattaforme PaaS, come Heroku o Azure App Service, semplificano il processo di deployment fornendo un ambiente gestito in cui è possibile caricare e gestire l'applicazione senza preoccuparsi dell'infrastruttura sottostante.

2. Migliori pratiche per la pubblicazione di applicazioni PHP

Durante la pubblicazione di un'applicazione PHP, è consigliabile seguire alcune migliori pratiche:

- Organizzazione del codice: Assicurarsi che il codice sia ben strutturato, organizzato e commentato per facilitarne la manutenzione e la collaborazione tra i membri del team.

- Utilizzo di un sistema di controllo di versione: Utilizzare un sistema di controllo di versione, come Git, per tenere traccia delle modifiche al codice e facilitare il lavoro di squadra. Inoltre, è consigliabile utilizzare un sistema di branching per separare le versioni di sviluppo da quelle di produzione.

- Configurazioni per ambienti diversi: Creare configurazioni specifiche per gli ambienti di sviluppo, test e produzione, in modo che l'applicazione si comporti correttamente in ogni ambiente.

- Automazione del deployment: Utilizzare strumenti di automazione, come script di deployment o strumenti di continuous integration/continuous deployment (CI/CD), per semplificare il processo di deployment e ridurre gli errori umani.

3. Considerazioni sulla sicurezza per la pubblicazione

Quando si pubblica un'applicazione PHP, è fondamentale prendere in considerazione la sicurezza. Alcune considerazioni importanti includono:

- Aggiornamenti regolari: Assicurarsi di utilizzare versioni aggiornate di PHP e delle librerie esterne per mitigare le vulnerabilità di sicurezza note.

- Validazione dei dati di input: Sempre validare e filtrare i dati di input per prevenire attacchi come SQL Injection o Cross-Site Scripting (XSS).

- Utilizzo di parametri preparati nelle query SQL: Per prevenire SQL Injection, utilizzare sempre parametri preparati o query parametrizzate invece

di concatenare direttamente i valori nelle query SQL.

- Protezione delle credenziali sensibili: Non includere credenziali sensibili, come password o chiavi API, nel codice sorgente. Utilizzare file di configurazione esterni o variabili d'ambiente per gestire tali informazioni.

- Filtraggio e validazione dei file caricati: Quando gli utenti possono caricare file nell'applicazione, è importante filtrare e validare attentamente tali file per prevenire l'esecuzione di codice dannoso o l'inserimento di file indesiderati.

- Protezione dalle vulnerabilità di sessione: Assicurarsi di utilizzare sessioni sicure e implementare le corrette politiche di gestione delle sessioni per prevenire attacchi come session hijacking o session fixation.

13 - Utilizzo dell'oggetto PDO

1. Introduzione all'oggetto PDO

L'oggetto PDO (PHP Data Objects) è una classe in PHP che fornisce un'interfaccia uniforme per l'accesso ai diversi database. Supporta diversi driver di database, consentendo di connettersi e interagire con il database in modo indipendente dal motore di archiviazione sottostante.

2. Connessione al database con PDO

Per connettersi a un database utilizzando PDO, è necessario specificare i dettagli di connessione, come l'host, il nome del database, l'utente e la password. Ecco un esempio di connessione a un database MySQL utilizzando PDO:

```
$host = 'localhost';
$dbname = 'mydatabase';
$username = 'myuser';
$password = 'mypassword';

try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
    // Altre operazioni sul database
} catch (PDOException $e) {
    echo "Errore nella connessione al database: " . $e->getMessage();
}
```

3. Esecuzione di query parametrizzate con PDO

Le query parametrizzate sono una pratica consigliata per prevenire attacchi di SQL Injection. Utilizzando i parametri nelle query, i valori inseriti dagli utenti vengono trattati come dati e non come parte della query stessa. Ecco un esempio di esecuzione di una query parametrizzata con PDO:

```
$sql = "SELECT * FROM users WHERE username = :username AND password = :password";
$stmt = $pdo->prepare($sql);

$username = 'john_doe';
$password = 'password123';

$stmt->bindParam(':username', $username);
$stmt->bindParam(':password', $password);

$stmt->execute();

$result = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

4. Gestione degli errori con PDO

PDO fornisce diverse opzioni per gestire gli errori durante l'esecuzione delle query. È possibile impostare l'attributo “PDO::ATTR_ERRMODE” su “PDO::ERRMODE_EXCEPTION” per generare un'eccezione PDOException in caso di errore. In questo modo, è possibile catturare l'eccezione e gestire gli errori in modo appropriato. Ecco un esempio:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    $stmt->execute();
} catch (PDOException $e) {
    echo "Errore durante l'esecuzione della query: " . $e->getMessage();
}
```

5. Sanitizzazione e prevenzione attacchi

La sanitizzazione dei dati di input è un'importante misura di sicurezza per prevenire attacchi come SQL Injection. Anche se l'utilizzo di query parametrizzate già fornisce una protezione significativa, è comunque consigliabile applicare ulteriori filtri e controlli sui dati di input.

- Filtraggio delle stringhe: Utilizzare le funzioni di filtraggio come “filter_var()” per rimuovere caratteri non validi o indesiderati dai dati di input.

```
$username = filter_var($_POST['username'], FILTER_SANITIZE_STRING);
```

- Validazione dei dati: Utilizzare le funzioni di validazione, ad esempio “filter_var()” con il filtro appropriato, per verificare che i dati di input rispettino i criteri desiderati.

```
$email = $_POST['email'];
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "L'indirizzo email non è valido";
}
```

6. Numero di righe coinvolte

Dopo l'esecuzione di una query con PDO, è possibile ottenere il numero di righe coinvolte dall'ultima operazione utilizzando il metodo “rowCount()” dell'oggetto statement. Ecco un esempio:

```
$stmt->execute();
$rowCount = $stmt->rowCount();
echo "Sono state coinvolte $rowCount righe nella query.";
```

14 - Invio di e-mail

1. Configurazione del server di posta

Prima di inviare e-mail tramite PHP, è necessario configurare correttamente il server di posta sul quale si desidera inviare i messaggi. Ciò implica l'individuazione dei parametri di configurazione corretti, come l'indirizzo del server SMTP, la porta, le credenziali di accesso e altre impostazioni specifiche del provider di posta o del server.

2. Invio di e-mail con PHPMailer

PHPMailer è una libreria molto popolare e potente per l'invio di e-mail tramite PHP. Per utilizzare PHPMailer, è necessario prima scaricare e includere la libreria nel tuo progetto. Ecco un esempio di invio di e-mail utilizzando PHPMailer:

```
require 'PHPMailer/PHPMailer.php';
require 'PHPMailer/SMTP.php';
// Crea un'istanza di PHPMailer
$mail = new PHPMailer\PHPMailer\PHPMailer();

// Configura le impostazioni del server di posta
$mail->isSMTP();
$mail->Host = 'smtp.example.com';
$mail->Port = 587;
$mail->SMTPAuth = true;
$mail->Username = 'your_username';
$mail->Password = 'your_password';

// Imposta il mittente e il destinatario
$mail->setFrom('sender@example.com', 'Mittente');
$mail->addAddress('recipient@example.com', 'Destinatario');

// Imposta il soggetto e il corpo del messaggio
$mail->Subject = 'Oggetto del messaggio';
$mail->Body = 'Contenuto del messaggio';

// Invia l'e-mail
if ($mail->send()) {
    echo 'E-mail inviata con successo';
} else {
    echo 'Errore durante l\'invio dell\'e-mail: ' . $mail->ErrorInfo;
}
```

3. Personalizzazione del messaggio e degli header

Con PHPMailer, è possibile personalizzare ulteriormente il messaggio e gli header dell'e-mail. Ad esempio, è possibile impostare il formato del testo (HTML o testo normale), aggiungere immagini o allegati, impostare gli header personalizzati, ecc. Ecco alcuni esempi:

```
// Imposta il formato del testo
$mail->isHTML(true); // Per inviare l'e-mail come HTML

// Aggiunge un'immagine incorporata nel messaggio
$mail->AddEmbeddedImage('path_to_image', 'image_cid', 'image_filename.jpg');

// Aggiunge un allegato
$mail->addAttachment('path_to_attachment', 'attachment_filename.pdf');

// Imposta gli header personalizzati
$mail->addCustomHeader('X-Custom-Header: Value');
```

4. Gestione degli allegati

Per allegare file alle e-mail inviate tramite PHPMailer, è possibile utilizzare il metodo “addAttachment()” come mostrato nell'esempio precedente. È possibile specificare il percorso del file e il nome con cui sarà visualizzato nell'e-mail. Puoi anche aggiungere più allegati chiamando ripetutamente il metodo “addAttachment()” per ogni file.

```
$mail->addAttachment('path_to_attachment1', 'attachment1_filename.pdf');
$mail->addAttachment('path_to_attachment2', 'attachment2_filename.jpg');
```

Assicurati di fornire i percorsi corretti ai file che desideri allegare.

15 - Esportazioni in vari formati

1. Esportare in PDF

L'esportazione in formato PDF è utile per generare documenti strutturati e formattati, adatti per la stampa o la visualizzazione. Per esportare dati o contenuti in formato PDF, puoi utilizzare librerie come TCPDF, Dompdf o FPDF. Ecco un esempio di esportazione in PDF utilizzando TCPDF:

```
require_once('tcpdf/tcpdf.php');

// Crea un'istanza di TCPDF
$pdf = new TCPDF();

// Imposta le informazioni del documento
$pdf->SetCreator('Nome del Creatore');
$pdf->SetAuthor('Nome dell'Autore');
$pdf->SetTitle('Titolo del Documento');
$pdf->SetSubject('Soggetto del Documento');

// Aggiungi una pagina al documento
$pdf->AddPage();

// Aggiungi il contenuto al documento
$pdf->SetFont('helvetica', '', 12);
$pdf->Cell(0, 10, 'Contenuto del documento PDF', 0, 1);

// Genera il file PDF
$pdf->Output('documento.pdf', 'D');
```

2. Esportare in Excel

L'esportazione in formato Excel è comunemente utilizzata per generare fogli di calcolo o report tabellari. Puoi utilizzare librerie come PhpSpreadsheet o PHPExcel per generare file Excel. Ecco un esempio di esportazione in Excel utilizzando PhpSpreadsheet:

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

use PhpOffice\PhpSpreadsheet\Spreadsheet;
use PhpOffice\PhpSpreadsheet\Writer\Xlsx;

// Crea un'istanza di Spreadsheet
$spreadsheet = new Spreadsheet();

// Ottieni il foglio di lavoro attivo
$sheet = $spreadsheet->getActiveSheet();

// Imposta i dati nell'Excel
$sheet->setCellValue('A1', 'Valore 1');
$sheet->setCellValue('B1', 'Valore 2');
$sheet->setCellValue('A2', 'Dato 1');
$sheet->setCellValue('B2', 'Dato 2');

// Crea un writer per l'esportazione in formato Excel
$writer = new Xlsx($spreadsheet);

// Salva il file Excel
$writer->save('file.xlsx');
```

3. Esportare in PNG e JPG

L'esportazione in formati di immagine come PNG e JPG è utile per generare grafici, grafici o immagini generate dinamicamente. Puoi utilizzare librerie come GD o Imagick per generare immagini in PHP. Ecco un esempio di esportazione in PNG utilizzando la libreria GD:

```
// Crea un'immagine vuota
$image = imagecreate(200, 200);

// Imposta il colore di sfondo
$background = imagecolorallocate($image, 255, 255, 255);

// Disegna un rettangolo rosso
$rectangleColor = imagecolorallocate($image, 255, 0, 0);
imagefilledrectangle($image, 50, 50, 150, 150, $rectangleColor);

// Esporta l'immagine in formato PNG
header('Content-Type: image/png');
imagepng($image, 'immagine.png');
imagedestroy($image);
```

```
<?php

        echo “<p>Fine!</p>”;

?>
```