

CFGS- Desenvolupament d'Aplicacions Web

Mòdul: Desenvolupament web en entorn servidor

Hores: 160

Codi: 0613.



Índex ->

Tema 1: Selecció d'arquitectures i eines de programació

1. Arquitectures en entorn servidor
2. Generació dinàmica de pàgines web
3. Llenguatges de programació i tecnologies associades en entorn servidor
4. Integració de PHP amb els llenguatges de marques
5. Integració amb els servidors web

Tema 2: Inserció de codi a pàgines web:

1. Introducció a PHP
2. Variables
3. Echo i print
4. Tipus de dades
5. Strings
6. Nombres
7. Càsting, matemàtiques i constants
8. Operadors
9. if..else
10. Bucles
11. Funcions
12. Arrays
13. Superglobals
14. Expressions regulars

Tema 3: PHP avançat

1. Formularis
2. Data i hora
3. Fitxers
4. Cookies
5. Sessions
6. Filtres
7. Funcions callback
8. JSON
9. Excepcions

Tema 4: OOP en PHP

1. OOP
2. Classes i objectes PHP
3. Constructor
4. Destructor
5. Modificador d'accés
6. Herència
7. Constants de classe
8. Classes abstractes
9. Interfícies
10. Característiques
11. Mètodes estàtics
12. Propietats estàtiques
13. Espais de noms
14. Iterables

Tema 5: PHP-MySQL

1. Base de dades MySQL

2. Connexió de PHP amb MySQL
3. Crea una base de dades
4. Crea una taula
5. Inserció de dades
6. Insereix múltiples
7. Select Data
8. where
9. Ordena per
10. Elimina dades
11. Actualitza dades
12. Límit de dades MySQL

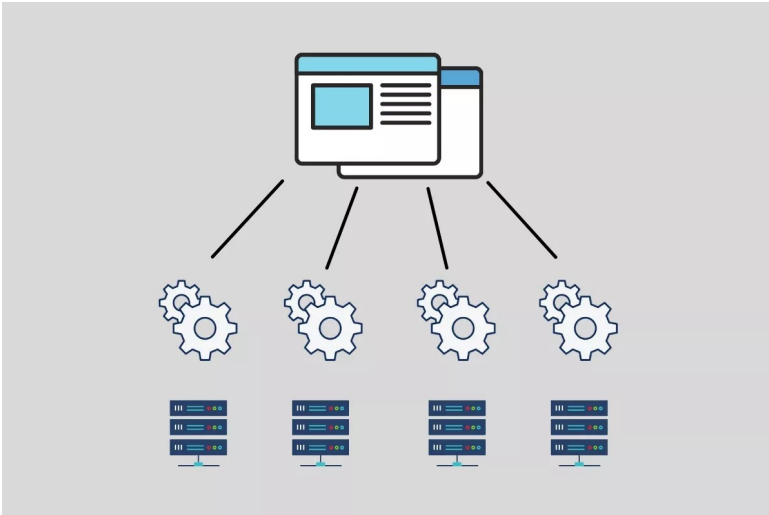
Tema 6: [PHP XML](#)

1. Analitzadors PHP XML
2. PHP SimpleXML Analitzador
3. PHP SimpleXML - Get
4. PHP XML Expat
5. PHP XML DOM

Tema 7: [PHP - AJAX](#)

1. Introducció a AJAX
2. AJAX PHP
3. AJAX Base de dades
4. AJAX XML
5. AJAX Cerca en directe
6. AJAX Enquesta

Tema 1: Selecció d'arquitectures i eines de programació



1. Arquitectures en entorn servidor
 2. Generació dinàmica de pàgines web
 3. Llenguatges de programació i tecnologies associades en entorn servidor
 4. Integració de PHP amb els llenguatges de marques
 5. Integració amb els servidors web
-

1. Arquitectures en entorn servidor

La selecció d'arquitectures i eines de programació per a entorns de servidor depèn de diversos factors, com ara les necessitats del projecte, les habilitats de l'equip, el rendiment requerit i altres consideracions tècniques i empresarials. Ací tens una visió general de les arquitectures i eines més comunes utilitzades en la programació en entorn servidor:

Arquitectures

1. Arquitectura Monolítica

- **Descripció:** Tot el codi de l'aplicació es troba en un únic projecte o servei.
- **Avantatges:** Simplicitat en el desplegament i el desenvolupament inicial, menys complexitat en la comunicació entre components.
- **Desavantatges:** Dificultat per escalar parts específiques de l'aplicació, desplegaments més complexos a mesura que l'aplicació creix.

2. Microserveis

- **Descripció:** L'aplicació es divideix en serveis petits i independents que es comuniquen entre ells.
- **Avantatges:** Facilitat d'escalabilitat, desplegaments més flexibles, millora en la mantenibilitat del codi.
- **Desavantatges:** Complexitat en la gestió de la comunicació entre serveis, necessitat de gestionar la consistència de les dades.

3. Serverless (Sense servidor)

- **Descripció:** Execució de funcions individuals en resposta a esdeveniments, sense necessitat de gestionar servidors.
- **Avantatges:** Escalabilitat automàtica, pagament només pels recursos utilitzats, menys manteniment d'infraestructura.
- **Desavantatges:** Limitacions en el temps d'execució, complexitat en depurar i provar.

Eines de Programació

1. Llenguatges de Programació

- **JavaScript (Node.js):** Popular per la seva velocitat i l'ecosistema vibrant de paquets.
- **Python:** Fàcil d'aprendre, amb una gran quantitat de biblioteques per a diferents necessitats.
- **Java:** Robusta i escalable, àmpliament utilitzada en entorns empresarials.
- **Ruby (Ruby on Rails):** Bona per a desenvolupament ràpid d'aplicacions web.
- **Go:** Alt rendiment i fàcil de compilar a binaris independents.

2. Frameworks i Llibreries

- **Express (Node.js):** Framework minimalista i flexible per a aplicacions web.
- **Django (Python):** Framework altament complet, inclou tot el necessari per desenvolupar aplicacions web.

- **Spring Boot (Java):** Framework àmpliament utilitzat per a aplicacions empresarials amb microserveis.
- **Flask (Python):** Microframework que permet crear aplicacions web amb gran flexibilitat.
- **Gin (Go):** Framework lleuger i ràpid per a aplicacions web.

3. Bases de Dades

- **SQL (Relacional):** MySQL, PostgreSQL, Oracle DB.
- **NoSQL (No Relacional):** MongoDB, Cassandra, Redis.
- **Bases de Dades en Memòria:** Redis, Memcached.

4. Plataformes de Contenedors

- **Docker:** Estandard per a contenidors, facilita el desplegament i la gestió d'aplicacions.
- **Kubernetes:** Orquestració de contenidors a gran escala.

5. Serveis Serverless

- **AWS Lambda:** Servei serverless d'Amazon.
- **Google Cloud Functions:** Servei serverless de Google.
- **Azure Functions:** Servei serverless de Microsoft.

6. Eines de DevOps i CI/CD

- **Jenkins:** Automatitza el procés de construcció, prova i desplegament.
- **GitLab CI/CD:** Integració contínua i desplegament continu amb GitLab.
- **CircleCI:** Plataforma de CI/CD per a desenvolupament ràpid i fiable.

Factors a Considerar en la Selecció

1. **Requisits del Projecte:** Complexitat, escalabilitat, requisits de rendiment.
2. **Habilitats de l'Equip:** Coneixements i experiència de l'equip en eines específiques.
3. **Costos:** Tant els costos directes com els de manteniment a llarg termini.
4. **Ecosistema:** Suport comunitari, biblioteques disponibles, eines complementàries.
5. **Integració:** Capacitat d'integrar-se amb altres sistemes i serveis existents.

Seleccionar la combinació adequada d'arquitectura i eines de programació és clau per l'èxit d'un projecte de desenvolupament de servidor. Cada decisió ha de basar-se en una avaluació acurada dels requisits del projecte i dels recursos disponibles.

2. Generació dinàmica de pàgines web

La generació dinàmica de pàgines web es refereix al procés mitjançant el qual el contingut d'una pàgina web es crea o es modifica en temps real, segons les necessitats o accions dels usuaris. A diferència de les pàgines web estàtiques, on el contingut és fix i no canvia fins que no es modifica manualment el codi, les pàgines dinàmiques s'actualitzen automàticament en funció de dades, interaccions de l'usuari o altres factors.

Components Clau

1. **Servidor web i Llenguatges de Programació del Costat del Servidor:**

- Els llenguatges de programació com PHP, Python (amb frameworks com Django o Flask), Ruby (amb Rails), Java (amb Spring), i Node.js es fan servir per generar contingut dinàmic al costat del servidor.
- Quan un usuari fa una sol·licitud a una pàgina web, el servidor executa un script que pot accedir a una base de dades, processar dades i generar HTML que s'envia de tornada al navegador de l'usuari.

2. Bases de Dades:

- Les bases de dades com MySQL, PostgreSQL, MongoDB, i altres són fonamentals per emmagatzemar i recuperar dades dinàmiques.
- Les sol·licituds de l'usuari sovint comporten consultes a la base de dades per a obtenir la informació necessària per mostrar la pàgina.

3. Llenguatges del Costat del Client:

- JavaScript és el llenguatge predominant per crear interactivitat al costat del client.
- Frameworks i biblioteques com React, Angular, Vue.js, i jQuery permeten crear interfícies d'usuari dinàmiques que poden actualitzar-se sense necessitat de recarregar tota la pàgina.

4. AJAX (Asynchronous JavaScript and XML):

- Permet que les pàgines web sol·liciten xicotetes quantitats de dades al servidor de manera asincrònica, sense haver de recarregar tota la pàgina.
- Això millora l'experiència de l'usuari, ja que les actualitzacions es fan de manera ràpida i fluida.

Exemples de Generació Dinàmica

1. Blogs i Sistemes de Gestió de Continguts (CMS):

- Plataformes com WordPress generen pàgines de manera dinàmica segons el contingut emmagatzemat a la base de dades.

2. Comerç Electrònic:

- Botigues en línia generen pàgines de producte, cistelles de compra i recomanacions de manera dinàmica segons les accions de l'usuari i l'inventari.

3. Aplicacions Web:

- Serveis com Gmail o Facebook utilitzen generació dinàmica per mostrar correus, notícies o altres continguts personalitzats en temps real.

Beneficis

- **Personalització:** Permet oferir una experiència personalitzada a cada usuari.
- **Actualització en Temps Real:** Les dades es poden actualitzar automàticament sense necessitat d'intervenció manual.

- **Interactivitat:** Millora l'experiència de l'usuari mitjançant interfícies més dinàmiques i reactives.

Consideracions

- **Rendiment:** La generació dinàmica pot ser més exigent en termes de recursos del servidor i temps de càrrega.
- **Seguretat:** Cal tenir en compte les vulnerabilitats com SQL injection, XSS (Cross-Site Scripting), i altres.

En resum, la generació dinàmica de pàgines web permet crear llocs més rics i interactius, adaptant-se a les necessitats i preferències dels usuaris en temps real. Utilitza una combinació de tecnologies del costat del servidor i del costat del client per aconseguir-ho.

3. Llenguatges de programació i tecnologies associades en entorn servidor

3.1 Llenguatges de Programació

JavaScript (Node.js)

Descripció: Node.js permet executar JavaScript al servidor, permetent el desenvolupament de backend amb el mateix llenguatge utilitzat en el frontend.

Tecnologies associades: Express.js, Koa.js, NestJS, Socket.IO.

Python

Descripció: Python és conegut per la seua senzillesa i llegibilitat, i és molt utilitzat en el desenvolupament web i de serveis al núvol.

Tecnologies associades: Django, Flask, FastAPI.

Java

Descripció: Java és un llenguatge robust i segur, utilitzat àmpliament en aplicacions empresarials a gran escala.

Tecnologies associades: Spring Framework, Hibernate, Apache Tomcat.

PHP

Descripció: PHP és un llenguatge àmpliament utilitzat per al desenvolupament web, especialment per aplicacions dinàmiques i sistemes de gestió de continguts.

Tecnologies associades: Laravel, Symfony, WordPress.

Ruby

Descripció: Ruby és conegut per la seva sintaxi elegant i el seu potent framework per al desenvolupament web.

Tecnologies associades: Ruby on Rails.

C# (ASP.NET)

Descripció: C# és un llenguatge de programació desenvolupat per Microsoft, i ASP.NET és el framework per a aplicacions web.

Tecnologies associades: .NET Core, Entity Framework, Blazor.

Go

Descripció: Go, també conegut com Golang, és un llenguatge dissenyat per a la programació de sistemes i aplicacions de xarxa.

Tecnologies associades: Gin, Echo, Revel.

Rust

Descripció: Rust és conegut per la seva seguretat i rendiment, sovint utilitzat en sistemes de baixa latència.

Tecnologies associades: Actix, Rocket.

3.2 Tecnologies i Eines

Bases de Dades

SQL: MySQL, PostgreSQL, Oracle Database.

NoSQL: MongoDB, Redis, Cassandra.

Servidors Web

Apache: Un dels servidors web més populars.

Nginx: Utilitzat per la seva alta eficiència i rendiment.

Contenidors i Orquestració

Docker: Per a la creació de contenidors.

Kubernetes: Per a l'orquestració de contenidors.

Plataformes al Núvol

AWS: Serveis al núvol d'Amazon.

Azure: Serveis al núvol de Microsoft.

Google Cloud: Serveis al núvol de Google.

Eines de Desenvolupament i CI/CD

Git: Sistema de control de versions.

Jenkins: Eina d'integració contínua.

Travis CI: Plataforma d'integració contínua.

Eines de Monitorització i Log Management

Prometheus: Sistema de monitorització.

Grafana: Plataforma d'analítica i monitorització.

ELK Stack: Elasticsearch, Logstash, Kibana per a la gestió de logs.

Aquestes són només algunes de les tecnologies i llenguatges més comuns en entorns servidor. La tria de les eines i llenguatges depèn sovint dels requisits específics del projecte, així com de les preferències i experiència de l'equip de desenvolupament.

4. Integració de PHP amb els llenguatges de marques

PHP és un dels llenguatges de programació més utilitzats per al desenvolupament web, conegut per la seva capacitat de generar i processar llenguatges de marques com HTML, XML i JSON. A continuació, es detallen exemples de com PHP s'integra amb aquests llenguatges de marques.

HTML

PHP és sovint utilitzat per generar contingut HTML dinàmic. Es poden inserir fragments de codi PHP directament dins d'arxius HTML per crear pàgines web interactives.

Exemple:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

    <meta charset="UTF-8">
    <title><?php echo $title; ?></title>
</head>
<body>
    <h1><?php echo $title; ?></h1>
    <p>Welcome, <?php echo $username; ?>!</p>
</body>
</html>

```

XML

PHP té funcions integrades per treballar amb XML, com SimpleXML i DOM. Aquestes funcions permeten tant la lectura com la generació de documents XML.

Lectura d'XML amb SimpleXML:

```

$xmlString = '<root><element key="value">Content</element></root>';
$xml = simplexml_load_string($xmlString);
echo $xml->element['key']; // Output: value
echo $xml->element; // Output: Content

```

Generació d'XML amb DOM:

```

$dom = new DOMDocument('1.0', 'UTF-8');
$root = $dom->createElement('root');
$dom->appendChild($root);

$element = $dom->createElement('element', 'Content');
$element->setAttribute('key', 'value');
$root->appendChild($element);

echo $dom->saveXML();

```

JSON

PHP té funcions natives per treballar amb JSON, com json_encode i json_decode, que permeten convertir dades de PHP a JSON i viceversa.

Codificació a JSON:

```

$data = array('key' => 'value', 'array' => array(1, 2, 3));
$json = json_encode($data);
echo $json; // Output: {"key":"value","array":[1,2,3]}

```

Decodificació de JSON:

```

$jsonString = '{"key":"value","array":[1,2,3]}';
$data = json_decode($jsonString, true);
echo $data['key']; // Output: value
print_r($data['array']); // Output: Array ( [0] => 1 [1] => 2 [2] => 3 )

```

Exemples Complets. Generació d'una pàgina HTML dinàmica amb PHP:

```

<?php
$title = "Home Page";
$username = "John Doe";
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title><?php echo $title; ?></title>
</head>
<body>
    <h1><?php echo $title; ?></h1>
    <p>Welcome, <?php echo $username; ?>!</p>
</body>
</html>

```

Processament d'una petició XML amb PHP:

```

<?php
$xmlString = file_get_contents('php://input');
$xml = simplexml_load_string($xmlString);

$response = new SimpleXMLElement('<response/>');
$response->addChild('status', 'success');
$response->addChild('received', $xml->asXML());

header('Content-Type: text/xml');
echo $response->asXML();

```

API RESTful que retorna JSON amb PHP:

```

<?php
header('Content-Type: application/json');

$data = array('key' => 'value', 'array' => array(1, 2, 3));
echo json_encode($data);

```

Aquests exemples mostren com PHP pot ser utilitzat per treballar amb HTML, XML i JSON, integrant-se perfectament amb aquests llenguatges de marques per proporcionar contingut dinàmic i interactiu en aplicacions web.

5. Integració de PHP amb servidors web

Integrar PHP amb servidors web és una tasca comuna en el desenvolupament web. PHP (Hypertext Preprocessor) és un llenguatge de script del costat del servidor dissenyat per al desenvolupament web. Els servidors web més comuns que s'utilitzen amb PHP són Apache i Nginx. A continuació es mostren les instruccions bàsiques per a la integració de PHP amb aquests servidors web.

5.1 Integració de PHP amb Apache

Instal·lació d'Apache:

```
sudo apt update
```

```
sudo apt install apache2
```

Instal·lació de PHP:

```
sudo apt install php libapache2-mod-php
```

Configuració d'Apache per utilitzar PHP:

Apache normalment detecta i configura automàticament PHP després de la instal·lació del mòdul libapache2-mod-php. Per assegurar-se que PHP està configurat correctament, podeu crear un fitxer de prova:

Creeu un fitxer anomenat info.php dins el directori /var/www/html/ amb el següent contingut:

```
<?php phpinfo(); ?>
```

Reinicieu Apache amb:

```
sudo systemctl restart apache2
```

Obriu el navegador i accediu a `http://<ip-del-teu-servidor>/info.php`. Hauríeu de veure una pàgina amb la configuració de PHP.

5.2 Integració de PHP amb Nginx

Instal·lació de Nginx:

```
sudo apt update
```

```
sudo apt install nginx
```

Instal·lació de PHP-FPM (FastCGI Process Manager):

```
sudo apt install php-fpm
```

Configuració de Nginx per utilitzar PHP:

Editeu el fitxer de configuració del lloc Nginx (normalment es troba a /etc/nginx/sites-available/default):

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;
    index index.php index.html index.htm;

    server_name _;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

```
location ~ \.php$ {  
    include snippets/fastcgi-php.conf;  
    fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;  
}  
  
location ~ /\.ht {  
    deny all;  
}  
}
```

Canvieu php7.4-fpm.sock pel fitxer de socket correcte si esteu utilitzant una altra versió de PHP.

Reinicieu Nginx amb:

```
sudo systemctl restart nginx
```

Proveu la configuració: Creeu un fitxer anomenat info.php com en Apache

Obriu el navegador i accediu a `http://<ip-del-teu-servidor>/info.php`. Hauríeu de veure una pàgina amb la configuració de PHP.

Seguint aquests passos, hauríeu de poder integrar PHP amb els servidors web Apache o Nginx. Això us permetrà executar scripts PHP en el vostre servidor web i desenvolupar aplicacions web dinàmiques.



-
1. [Introducció a PHP](#)
 2. [Variables](#)
 3. [Echo i print](#)
 4. [Tipus de dades](#)
 5. [Strings](#)
 6. [Nombres](#)
 7. [Càsting, matemàtiques i constants](#)
 8. [Operadors](#)
 9. [if..else](#)
 10. [Bucles](#)
 11. [Funcions](#)
 12. [Arrays](#)
 13. [Superglobals](#)
 14. [Expressions regulars](#)
-

1. Introducció a PHP

El codi PHP s'executa en el servidor.

Coses que ja hauries de saber abans de continuar, ha de tenir una comprensió bàsica de: *HTML, CSS i JavaScript*

Què és el PHP?

PHP és un acrònim de "PHP: Hypertext Preprocessor".

PHP és un llenguatge de scripting de codi obert àmpliament utilitzat.

Els scripts PHP s'executen en el servidor.

PHP és gratuït per descarregar i utilitzar i és un llenguatge molt interessant i popular.

Està present en moltes de les més grans i popular aplicacions web.

És, probablement, la millor opció per a començar en programació backend

Què és un arxiu PHP?

Els arxius PHP poden contenir text, HTML, CSS, JavaScript i codi PHP. El codi PHP s'executa en el servidor, i el resultat es retorna al navegador com a HTML pla. Els arxius PHP tenen extensió ".php"

Què pot fer PHP?

- PHP pot generar contingut dinàmic de pàgines
- PHP pot crear, obrir, llegir, escriure, esborrar i tancar arxius en el servidor.
- PHP pot recopilar dades de formularis
- PHP pot enviar i rebre cookies
- PHP pot afegir, eliminar, modificar les dades a la base de dades
- PHP es pot utilitzar per controlar l'accés a l'usuari.
- PHP pot xifrar les dades

Amb PHP no et limites només a una eixida HTML, podeu tenir una eixida d'imatges o arxius PDF. També podeu crear una eixida de qualsevol text, com XHTML i XML.

Per què PHP?

- PHP funciona en diverses plataformes (Windows, Linux, Unix, Mac OS X, etc.)
- PHP és compatible amb gairebé tots els servidors utilitzats actualment (Apache, IIS, etc.)

- PHP suporta una àmplia gamma de bases de dades
- PHP és gratuït. Descarregueu-vos-ho del recurs oficial PHP: www.php.net
- PHP és fàcil d'aprendre i funciona de manera eficient en el costat del servidor.

Instal·leu PHP al vostre PC

Si el vostre servidor no suporta PHP, heu de:

- Instal·lar un servidor web
- Instal·lar PHP
- Instal·lar una base de dades, com MySQL

Sintaxi bàsica de PHP

Un script PHP s'executa en el servidor, i el resultat d'HTML és enviat de nou al navegador.

Un script PHP es pot col·locar en qualsevol lloc del document. Un script PHP comença amb `<?php` i acaba amb `?>`

```
<?php
// PHP code goes here
?>
```

L'extensió d'arxiu per defecte per a arxius PHP és ".php". Un arxiu PHP normalment conté etiquetes HTML, i un codi de scripting PHP.

A continuació, tenim un exemple d'un simple arxiu PHP, amb un script PHP que utilitza una funció incorporada en PHP "echo" Per a l'eixida del text "Hola món!" en una pàgina web:

```
<!DOCTYPE html>
<html>
<body>
    <h1>My first PHP page</h1>

    <?php
    echo "Hello World!";
    ?>
</body>
</html>
```

Nota: Les declaracions de PHP acaben amb un semicolon (;)

Nota: PHP, pel que fa a paraules clau (per exemple. if, else, while, echo, etc.), classes, funcions, i les funcions definides per l'usuari, no és sensible a majúscules i minúscules. ECHO és el mateix que echo.

Nota: No obstant això, tots els noms variables són sensibles a majúscules i minúscules.

Comentaris en PHP

Un comentari en codi PHP és una línia que no s'executa com a part de la Programa. El seu únic objectiu és ser llegit per algú que està mirant el codi per a una millor comprensió del codi.

PHP suporta diverses maneres de comentar:

```
// This is a single-line comment
# This is also a single-line comment
/* This is a
multi-line comment */
```

2. Variables

Les variables són "contenidors" per emmagatzemar informació.

Creació (Declaració) PHP Variables

En PHP, una variable comença amb la \$, seguit del nom de la variable:

```
$x = 5;
$y = "John"
```

Nota: Quan assigneu un valor de text a una variable, poseu cites al voltant del valor.

Nota: A diferència d'altres llenguatges de programació, PHP no té cap ordre per declarar una variable. Es crea en el moment en què li assignes un valor.

Penseu en les variables com a contenidors per emmagatzemar dades.

Regles de les variables PHP:

- Una variable comença amb el \$, seguit del nom de la variable
- Un nom variable ha de començar amb una lletra o el caràcter subratllat.
- Un nom variable no pot començar amb un número.
- Un nom variable només pot contenir caràcters alfa-númericos i subratllar (A-z, 0-9 i _)

Els noms variables són de cas-sensibles (\$age i \$AGE són dues variables diferents)

El següent exemple mostrarà com a eixida text + una variable:

```
$txt = "W3Schools.com";
echo "I love $txt!";
```

El següent exemple produirà la mateixa eixida que l'exemple anterior:

```
$txt = "W3Schools.com";  
echo "I love " . $txt . "!";
```

El següent exemple produirà la suma de dues variables:

```
$x = 5;  
$y = 4;  
echo $x + $y;
```

PHP és un llenguatge no tipat. En l'exemple anterior, noteu que no havíem de dir a PHP quin tipus de dades és la variable.

PHP associa automàticament un tipus de dades a la variable, depenent del seu valor. Com que els tipus de dades no s'estableixen en un sentit estricte, es poden fer coses com afegir un string a un enter sense causar un error.

En PHP 7, s'hi van afegir declaracions tipus. Això dóna una opció per especificar el tipus de dades que s'espera en declarar una funció, i en permetre el requisit estricte, es llançarà un "Fatal Error" en un tipus desajustat.

Tipus de variables

PHP no té cap ordre per declarar una variable, i el tipus de dades depèn del valor de la variable.

```
$x = 5;          // $x is an integer  
$y = "John";    // $y is a string  
echo $x;  
echo $y;
```

PHP dona suport als següents tipus de dades: String, Integer, Float, Boolean, Array, Object, NULL i Resource.

Per obtenir el tipus de dades d'una variable, utilitzeu el `var_dump()`. La funció `var_dump()` retorna el tipus de dades i el valor. Funciona així:

```
<!DOCTYPE html>  
<html>  
<body>  
  
    <?php  
        var_dump($x);  
        var_dump(5);  
        var_dump("John");  
        var_dump(3.14);  
        var_dump(true);  
        var_dump([2, 3, 56]);  
        var_dump(NULL);  
    ?>  
  
</body>  
</html>
```

Retorna: NULL int(5) string(4) "John" float(3.14) bool(true) array(3) { [0]=> int(2) [1]=> int(3) [2]=> int(56) } NULL

Els string

```
$x = "John";  
echo $x;
```

Assignació de valors múltiples

Es pot assignar el mateix valor a múltiples variables en una sola línia:

```
$x = $y = $z = "Fruit";
```

Abast de les variables

En PHP, les variables es poden declarar en qualsevol lloc. L'abast d'una variable és la part del script on la variable pot ser referenciada/utilitzada.

PHP té tres abasts de variables diferents: Local, Global i Estàtic

Abast global i local

Una variable declarada fora d'una funció té una ESCOPE GLOBAL i només pot ser accedida a l'exterior d'una funció:

```
<!DOCTYPE html>  
<html>  
<body>  
<?php  
    $x = 5; // global scope  
  
    function myTest() {  
        // using x inside this function will generate an error  
        echo "<p>Variable x inside function is: $x</p>";  
    }  
    myTest();  
  
    echo "<p>Variable x outside function is: $x</p>";  
?>  
</body>  
</html>
```

Una variable declarada dins d'una funció té una ESCOPE LOCAL i només pot S'hi accedeix dins d'aquesta funció:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
    function myTest() {  
        $x = 5; // local scope  
        echo "<p>Variable x inside function is: $x</p>";  
    }  
    myTest();
```

```

        // using x outside the function will generate an error
        echo "<p>Variable x outside function is: $x</p>";
?>
</body>
</html>

```

Es poden tenir variables locals amb el mateix nom en diferents funcions. perquè les variables locals només es reconeixen per la funció en què es troben Declarat.

PHP La paraula clau global

La paraula clau global s'utilitza per accedir a una variable global des de dins d'una funció.

```

<!DOCTYPE html>
<html>
<body>
<?php
    $x = 5;
    $y = 10;

    function myTest() {
        global $x, $y;
        $y = $x + $y;
    }

    myTest(); // run function
    echo $y; // output the new value for variable $y
?>
</body>
</html>

```

PHP també emmagatzema totes les variables globals en un array anomenada \$GLOBALS[index]. El index té el nom de la variable. Aquest array també és accessible des de dins de les funcions i es pot utilitzar per actualitzar directament les variables globals.

```

<!DOCTYPE html>
<html>
<body>
<?php
    $x = 5;
    $y = 10;

    function myTest() {
        $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
    }

    myTest();
    echo $y;
?>
</body>
</html>

```

PHP La paraula clau estàtica

Normalment, quan una funció finalitza (ja s'ha executat), totes les seues variables s'eliminen. No obstant això, de vegades volem que una variable local no s'esborri. La necessitem per a altra feina.

Per fer-ho, utilitzeu la paraula clau static quan es declara per primera vegada la variable:

```
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}
```

```
myTest();
myTest();
myTest();
```

Cada vegada que es crida a la funció, aquesta variable encara tindrà la la informació que contenia des de l'última vegada que es va anomenar la funció.

3. Echo i print

En PHP, hi ha dues maneres bàsiques d'obtenir eixida: echo i print. Echo i print és més o menys el mateix.

Les diferències són xicotetes. Echo no té cap valor de retorn mentre print té un valor de retorn d'1 si tot va bé.

La declaració d'echo es pot utilitzar amb o sense parèntesis: echoo echo(). El text pot contenir el marcatge HTML

```
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
```

El següent exemple mostra com a eixida text i variables amb un echo

```
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
```

Print

La declaració de print es pot utilitzar amb o sense els parèntesis: printo print().

El següent exemple mostra eixida de text amb el print

```
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
```

El següent exemple mostra com a eixida text i variables

```
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;
```

```
print "<h2>" . $txt1 . "</h2>";  
print "Study PHP at " . $txt2 . "<br>";  
print $x + $y;
```

4. Tipus de dades

Les variables poden emmagatzemar dades de diferents tipus. PHP és compatible amb els següents tipus de dades:

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

Obtenir el tipus de dades

Podeu obtenir el tipus de dades de qualsevol objecte mitjançant `var_dump()`. Funciona. La funció `var_dump()` retorna el tipus de dades i el valor:

```
$x = 5;  
var_dump($x);
```

String

String és una cadena de caràcters com "Hola món!"

Una cadena pot ser qualsevol text dins de les cites. Podeu utilitzar les cites individuals o dobles:

```
$x = "Hello world!";  
$y = 'Hello world!';  
  
var_dump($x);  
echo "<br>";  
var_dump($y);
```

Integer

Un tipus de dades enter és un nombre no decimal entre -2.147.483.648 i 2.147.483.647.

Regles per a enters:

- Un enter ha de tenir almenys un dígit
- Un enter no ha de tenir un punt decimal
- Un enter pot ser positiu o negatiu.
- Els enters es poden especificar en: decimal (base 10), hexadecimal (base 16), octal (base 8), o binaris (base 2) notació

En el següent exemple `$x` és un enter. La funció `var_dump()` retorna el tipus de dades i el valor:

```
$x = 5985;  
var_dump($x);
```

Float

Un float (nombre de punt flotant) és un nombre amb un punt decimal o un nombre en forma exponencial.

En el següent exemple \$x és un float.

```
$x = 10.365;  
var_dump($x);
```

Boolean

Un booleà representa dos possibles estats: TRUE o FALSE.

```
$x = true;  
var_dump($x);
```

Array

Un array emmagatzema múltiples valors en una sola variable. En el següent exemple \$cars és un array.

```
$cars = array("Volvo", "BMW", "Toyota");  
var_dump($cars);
```

Classes

Les classes i els objectes són els dos aspectes principals de la programació orientada a objectes (OOP).

Una classe és una plantilla per a objectes, i un objecte és una instància de classe.

Quan es creen els objectes individuals, hereten totes les propietats i comportaments de la classe, però cada objecte tindrà valors diferents per a les propietats.

Assumim que tenim una classe que es diu Car, pot tenir propietats com el model, el color, etc. Podem definir variables com \$model, \$color i , potser, altres.

Quan es creen els objectes individuals (Volvo, BMW, Toyota, etc.) heretar totes les propietats i comportaments de la classe, però cada objecte té diferents valors per a les propietats.

Si es crea una funció __construct(), aquesta s'executarà quan es cree un objecte a partir d'una classe.

```
class Car {  
    public $color;  
    public $model;  
    public function __construct($color, $model) {  
        $this->color = $color;  
        $this->model = $model;  
    }  
}
```

```

    public function message() {
        return "My car is a " . $this->color . " " . $this->model . "!";
    }
}

$myCar = new Car("red", "Volvo");
var_dump($myCar);

```

NULL Value

Null és un tipus especial de dades que només pot tenir un valor: NULL. Una variable de tipus NULL és una variable que no té cap valor assignat.

Consell: Si una variable es crea sense un valor, és Assigna automàticament un valor de NULL.

```

$x = "Hello world!";
$x = null;
var_dump($x);

```

Canviar el tipus de dades

Si assignes un valor enter a una variable, el tipus serà automàticament un enter. Si assignes un string a la mateixa variable, el tipus canviarà a string:

```

$x = 5;
var_dump($x);

$x = "Hello";
var_dump($x);

```

Si voleu canviar el tipus de dades d'una variable existent, però no canviar el valor, es pot utilitzar el càsting. El càsting permet canviar el tipus de dades sobre les variables:

Exemple

```

$x = 5;
$x = (string) $x;
var_dump($x);

```

Resource

El tipus de recurs especial no és un tipus de dades real. És l'emmagatzematge d'una referència a funcions i recursos externs a PHP.

Un exemple comú de l'ús del tipus de dades de recursos és una crida a base de dades.

5. Strings

Un string és una seqüència de caràcters, com "Hola món!". Els string en PHP van entre cita doble o simple.

```

echo "Hello";
echo 'Hello';

```

Cites dobles o individuals?

Nota: Les quotes dobles processen caràcters especials. Les cites individuals no. Prova aquests dos exemples i ho tindràs clar.

```
$x = "John";  
echo "Hello $x";
```

```
$x = "John";  
echo 'Hello $x';
```

Longitud d'string

El PHP la funció strlen() retorna la longitud d'un string

```
echo strlen("Hello world!");
```

Word Count

La funció str_word_count() compta el Nombre de paraules en un string.

```
echo str_word_count("Hello world!");
```

Cerca de text dins d'un string

La funció strpos() busca un text específic dins d'una cadena. Si es troba una ocurrència, la funció retorna la posició de caràcter de la ocurrència. Si no es troba cap ocurrència, tornarà FALSE.

Busca el text "món" en la cadena "Hola món!":

```
echo strpos("Hello world!", "world");
```

Upper Case

```
$x = "Hello World!";  
echo strtoupper($x);
```

Lower Case

```
$x = "Hello World!";  
echo strtolower($x);
```

Replace String

```
$x = "Hello World!";  
echo str_replace("World", "Dolly", $x);
```

Reverse a String

```
$x = "Hello World!";  
echo strrev($x);
```

Remove Whitespace

```
$x = " Hello World! ";  
echo trim($x);
```

Convert String into Array

El primer paràmetre de la funció `explode()` representa el separador. El searador especifica on dividir l'string

Nota: és necessari el separador.

```
$x = "Hello World!";
$y = explode(" ", $x);

//Use the print_r() function to display the result:
print_r($y);

/*
Result:
Array ( [0] => Hello [1] => World! )
*/
```

String concatenation

Per concatenar, o combinar, dos strings es poden utilitzar l'operador `(.)`:

```
$x = "Hello";
$y = "World";
$z = $x . $y;
echo $z;
```

El resultat de l'exemple anterior és HelloWorld, sense Espai entre les dues paraules.

Podeu afegir un caràcter espacial com ara espai així:

```
$x = "Hello";
$y = "World";
$z = $x . " " . $y;
echo $z;
```

Una manera més fàcil i millor és utilitzant el poder de les quotes dobles.

Al voltant de les dues variables en dobles cites amb un espai blanc entre elles. L'espai blanc també estarà present en el resultat:

```
$x = "Hello";
$y = "World";
$z = "$x $y";
echo $z;
```

substr()

Podeu retornar una sub-cadena de caràcters utilitzant la funció `substr()`.

Especifiqueu l'índex d'inici i el nombre de caràcters que voleu tornar.

```
$x = "Hello World!";
echo substr($x, 6, 5);
```

substr fins al final

En deixar fora el paràmetre de llargada, la sub-cadena anirà fins al final:

```
$x = "Hello World!";  
echo substr($x, 6);
```

subrt des del final

Utilitzeu índex negatius per iniciar la sub-cadena des del final de l'string:

```
$x = "Hello World!";  
echo substr($x, -5, 3);
```

Caràcter Escap

Per inserir caràcters il·legals en una cadena, utilitzeu el caràcter d'escapament. Heu de posar \ i a continuació, el caràcter que voleu inserir.

```
$x = "We are the so-called \"Vikings\" from the north.";
```

Per solucionar aquest problema, utilitzeu el caràcter d'escapament \

```
$x = "We are the so-called \"Vikings\" from the north.";
```

Caràcters d'escapament

Altres caràcters d'escapament utilitzats en PHP:

Code Result

\'	Single Quote
\"	Double Quote
\\$	PHP variables
\n	New Line
\r	Carriage Return
\t	Tab
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

6. Nombres

Hi ha tres tipus numèrics principals en PHP: Integer, Float i Number Strings

A més, PHP té dos tipus de dades més utilitzats per a nombres: Infinity i NaN

Les variables de tipus numèric es creen quan s'assigna un valor a elles:

```
$a = 5;  
$b = 5.34;  
$c = "25";
```

Integer

2, 256, -256, 10358, -179567 són enters. Un enter és un nombre sense cap part decimal.

Un tipus de dades enter és un nombre no decimal entre -2147483648 i 2147483647 en sistemes de 32 bits, i entre -9223233703685807570808 i 9223372036854775807 en sistemes de 64 bits. Un valor més gran (o més baix) que aquest, s'emmagatzemarà com a float, perquè supera el límit d'un enter.

Nota: Una altra cosa important a saber és que encara que $4 * 2,5$ és 10 i el resultat s'emmagatzemat com a float, perquè un dels operands és un float.

PHP té les següents constants predefinides per a enters:

- PHP_INT_MAX- El major enter suportat
- PHP_INT_MIN- El més xicotet enter suportat
- PHP_INT_SIZE La mida d'un enter en bytes

PHP té les següents funcions per comprovar si el tipus de variable és enter: `is_int()`

Comproveu si el tipus d'una variable és enter:

```
$x = 5985;  
var_dump(is_int($x));
```

```
$x = 59.85;  
var_dump(is_int($x));
```

Float

Un float és un nombre amb un punt decimal o un nombre en forma exponencial.

2.0, 256.4, 10.358, 7.64E+5, 5,56E-5 són tots float.

El tipus de dades flotants normalment pot emmagatzemar un valor fins a 1.7976931348623E+308 (dependent de la plataforma), i tenir una precisió màxima de 14 dígit.

PHP té les següents constants predefinides per a float:

- PHP_FLOAT_MAX- El major nombre de punts flotants representables
- PHP_FLOAT_MIN- El menor nombre de punts flotants positius representables més petit
- PHP_FLOAT_DIG El nombre de dígit decimals que es poden arrodonir en un flotar i tornar sense pèrdua de precisió
- PHP_FLOAT_EPSILON- El nombre positiu representable més xicotet

PHP té les següents funcions per comprovar si el tipus de variable és Flo flotant:

- `is_float()`
- `is_double()`- Alias de `is_float()`

Comproveu si el tipus de variable és flotant:

```
$x = 10.365;  
var_dump(is_float($x));
```

Infinity

Un valor numèric que és més gran que PHP_FLOAT_MAX Es considera infinit.

PHP té les següents funcions per comprovar si un valor numèric és finit o Infinit:

```
is_finite()
is_infinite()
```

NaN

NaN Es fa servir per a operacions matemàtiques impossibles.

PHP té les següents funcions per comprovar si un valor no és un número:

```
is_nan()
```

```
<!DOCTYPE html>
<html>
<body>

<?php
// Invalid calculation will return a NaN value
$x = acos(8);
var_dump($x);
?>
// retorna float(NAN)
</body>
</html>
```

Numerical Strings

La funció `is_numeric()` es pot utilitzar per a saber si una variable és numèrica. La funció torna cert si la variable és un nombre.

```
$x = 5985;
var_dump(is_numeric($x));

$x = "5985";
var_dump(is_numeric($x));
$x = "59.85" + 100;
var_dump(is_numeric($x));

$x = "Hello";
var_dump(is_numeric($x));
```

De vegades cal canviar una variable d'un tipus de dades en un altre. I de vegades es vol una variable per tenir un tipus de dades específic. Això es pot fer amb el càsting. Canviar el tipus de dades

7. Càsting, matemàtiques i constants

El càsting en PHP es fa amb aquestes declaracions:

- (string)- Conversió a dades tipus String
- (int)- Conversió a dades tipus Integer
- (float)- Conversió a dades tipus Float
- (bool)- Conversió a dades tipus Boolean
- (array)- Conversió a dades tipus Array

(object)- Conversió a data tipus Objecte

(unset)- Conversió a dades tipus NULL

Cast a String

```
$a = 5;           // Integer
$b = 5.34;        // Float
$c = "hello";     // String
$d = true;        // Boolean
$e = NULL;        // NULL
```

```
$a = (string) $a;
$b = (string) $b;
$c = (string) $c;
$d = (string) $d;
$e = (string) $e;
```

//To verify the type of any object in PHP, use the var_dump() function:

```
var_dump($a);
var_dump($b);
var_dump($c);
var_dump($d);
var_dump($e);
```

Transformació d'objectes en Arrays:

```
class Car {
    public $color;
    public $model;
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }
    public function message() {
        return "My car is a " . $this->color . " " . $this->model . "!";
    }
}
```

```
$myCar = new Car("red", "Volvo");
```

```
$myCar = (array) $myCar;
var_dump($myCar);
```

Cast a object

Per passar a object, utilitzeu la declaració (object):

```
$a = 5;           // Integer
$b = 5.34;        // Float
$c = "hello";     // String
$d = true;        // Boolean
$e = NULL;        // NULL
```

```
$a = (object) $a;
$b = (object) $b;
$c = (object) $c;
$d = (object) $d;
$e = (object) $e;
```

En convertir-se en objectes, la majoria dels tipus de dades es converteixen en un objecte amb una propietat, anomenada "escalar", amb el valor corresponent.

Els valors NULL es converteixen en un objecte buit.

Els arrays indexats es converteixen en objectes amb el número d'índex com a nom de propietat i el valor com a valor de propietat.

Els arrays associatius es converteixen en objectes amb les claus com a noms de propietat i valors com a valors de propietat.

```
$a = array("Volvo", "BMW", "Toyota"); // indexed array
$b = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); // associative array
```

```
$a = (object) $a;
$b = (object) $b;
```

Cast a NULL

```
$a = 5;           // Integer
$b = 5.34;        // Float
$c = "hello";     // String
$d = true;        // Boolean
$e = NULL;        // NULL
```

```
$a = (unset) $a;
$b = (unset) $b;
$c = (unset) $c;
$d = (unset) $d;
$e = (unset) $e;
```

Matemàtiques

PHP té un conjunt de funcions matemàtiques que li permet realitzar tasques matemàtiques amb nombres.

Funció **pi()**: retorna el valor de PI:

Funcions **min()** i **max()**: Les funcions es poden utilitzar per trobar el valor més baix o més alt en una llista d'arguments.

```
echo(min(0, 150, 30, 20, -8, -200));
echo(max(0, 150, 30, 20, -8, -200));
```

Funció **abs()**: La funció retorna el valor absolut (positiu) d'un nombre:

sqrt()

round()

El rand()

Per obtenir més control sobre el nombre aleatori, es pot afegir els paràmetres opcionals de min i màxim per especificar l'enter més baix i l'enter més alt a retornar.

Per exemple, si voleu un enter aleatori entre 10 i 100 (inclusiu), ús **rand(10, 100):**

```
echo(rand(10, 100));
```

Constants

Les constants són com variables, excepte que una vegada que es defineixen. No es poden canviar ni redefinir.

Una constant és un identificador (nom) per a un valor simple. El valor no pot ser canviat.

Un nom constant vàlid comença amb una lletra o subratllat (sense un signe de \$ abans el nom constant).

Nota: A diferència de les variables, les constants són automàticament globals a través de tot el programa.

Per a crear una constant, utilitzeu la funció `define()`

`define(name, value, case-insensitive);`

Paràmetres:

`name`Nom: especifica el nom de la constant

`value`Valor: especificar el valor de la constant

`case-insensitive`: especifica si el nom de la constant ha de ser case insensible

Crear una constant amb case-sensitive:

```
define("GREETING", "Welcome to W3Schools.com!");  
echo GREETING;
```

Crear una constant amb case-insensitive:

```
define("GREETING", "Welcome to W3Schools.com!", true);  
echo greeting;
```

const keyword

També es pot crear una constant utilitzant la paraula clau `const`.

```
const MYCAR = "Volvo";  
echo MYCAR;
```

Arrays de constants

Des de PHP7, es pot crear una constant d'Array utilitzant la funció `define()`.

```
define("cars", [  
    "Alfa Romeo",  
    "BMW",  
    "Toyota"  
]);  
echo cars[0];
```

Constants predefinides de PHP

PHP té nou constants predefinides que canvien de valor depenent d'on s'utilitzen, i per això s'anomena "constants màgiques".

Aquestes constants màgiques s'escriuen amb un guió baix doble al principi i al final, excepte la constant `ClassName::class`.
Constants màgiques

Ací teniu les constants màgiques, amb descripcions i exemples:

Constant	Description
<code>__CLASS__</code>	If used inside a class, the class name is returned.
<code>__DIR__</code>	The directory of the file.
<code>__FILE__</code>	The file name including the full path.
<code>__FUNCTION__</code>	If inside a function, the function name is returned.
<code>__LINE__</code>	The current line number.
<code>__METHOD__</code>	If used inside a function that belongs to a class, both class and function name is returned.
<code>__NAMESPACE__</code>	If used inside a namespace, the name of the namespace is returned.
<code>__TRAIT__</code>	If used inside a trait, the trait name is returned.
<code>ClassName::class</code>	Returns the name of the specified class and the name of the namespace, if any.

Nota:

Les constants màgiques no distingeixen entre majúscules i minúscules, és a dir `__LINE__` retorna el mateix que `__line__`.

8. Operadors

Els operadors s'utilitzen per a realitzar operacions sobre variables i valors. PHP divideix els operadors en els següents grups:

- Operadors aritmètics
- Operadors d'assignació
- Operadors de comparació
- Operadors d'increment/disminució
- Operadors lògics
- Operadors de cadena
- Operadors d'array
- Operadors d'assignació condicional

Operadors aritmètics

Els operadors aritmètics PHP s'utilitzen amb valors numèrics per realitzar operacions aritmètiques habituals, com sumes, restes, multiplicacions, etc.

Operator	Name	Example	Result
<code>+</code>	Addition	<code>\$x + \$y</code>	Sum of <code>\$x</code> and <code>\$y</code>
<code>-</code>	Subtraction	<code>\$x - \$y</code>	Difference of <code>\$x</code> and <code>\$y</code>
<code>*</code>	Multiplication	<code>\$x * \$y</code>	Product of <code>\$x</code> and <code>\$y</code>
<code>/</code>	Division	<code>\$x / \$y</code>	Quotient of <code>\$x</code> and <code>\$y</code>
<code>%</code>	Modulus	<code>\$x % \$y</code>	Remainder of <code>\$x</code> divided by <code>\$y</code>
<code>**</code>	Exponential	<code>\$x ** \$y</code>	Result of raising <code>\$x</code> to the <code>\$y</code> 'th power

Operadors d'assignació

Els operadors d'assignació PHP s'utilitzen amb valors numèrics per escriure un valor a una variable.

L'operador bàsic d'assignació en PHP és "=". Vol dir que l'operand esquerre s'estableix en el valor de l'expressió d'assignació de la dreta.

Assignment	Same as...	Description
<code>x = y</code>	<code>x = y</code>	The left operand gets set to the value of the right
<code>x += y</code>	<code>x = x + y</code>	Addition
<code>x -= y</code>	<code>x = x - y</code>	Subtraction
<code>x *= y</code>	<code>x = x * y</code>	Multiplication
<code>x /= y</code>	<code>x = x / y</code>	Division
<code>x %= y</code>	<code>x = x % y</code>	Modulus

Operadors de comparació PHP

Els operadors de comparació PHP s'utilitzen per comparar dos valors (número o cadena):

Operator	Name	Example	Result
<code>==</code>	Equal	<code>\$x == \$y</code>	Returns true if \$x is equal to \$y
<code>===</code>	Identical	<code>\$x === \$y</code>	Returns true if \$x is equal to \$y, and they are of the same type
<code>!=</code>	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
<code><></code>	Not equal	<code>\$x <> \$y</code>	Returns true if \$x is not equal to \$y
<code>!==</code>	Not identical	<code>\$x !== \$y</code>	Returns true if \$x is not equal to \$y, or they are not of the same type
<code>></code>	Greater than	<code>\$x > \$y</code>	Returns true if \$x is greater than \$y
<code><</code>	Less than	<code>\$x < \$y</code>	Returns true if \$x is less than \$y
<code>>=</code>	Greater than or =	<code>\$x >= \$y</code>	Returns true if \$x is greater than or equal to \$y
<code><=</code>	Less than or =	<code>\$x <= \$y</code>	Returns true if \$x is less than or equal to \$y
<code><=></code>	Spaceship	<code>\$x <=> \$y</code>	Returns an integer less than, equal to, or greater than zero, depending on if \$x is less than, equal to, or greater than \$y.

Operadors d'increment/decrement

Els operadors d'increment de PHP s'utilitzen per incrementar el valor d'una variable.

Els operadors de disminució de PHP s'utilitzen per disminuir el valor d'una variable.

Operator	Same as...	Description
<code>++\$x</code>	Pre-increment	Increments \$x by one, then returns \$x
<code>\$x++</code>	Post-increment	Returns \$x, then increments \$x by one
<code>--\$x</code>	Pre-decrement	Decrements \$x by one, then returns \$x
<code>\$x--</code>	Post-decrement	Returns \$x, then decrements \$x by one

Operadors lògics

Els operadors lògics PHP s'utilitzen per combinar declaracions condicionals.

Operator	Name	Example	Result
<code>and</code>	And	<code>\$x and \$y</code>	True if both \$x and \$y are true
<code>or</code>	Or	<code>\$x or \$y</code>	True if either \$x or \$y is true

xor	Xor	<code>\$x xor \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true, but not both
&&	And	<code>\$x && \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
	Or	<code>\$x \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
!	Not	<code>!\$x</code>	True if <code>\$x</code> is not true

Operadors de cadena

PHP té dos operadors dissenyats especialment per a cadenes.

Operator	Name	Example	Result
.	Concatenation	<code>\$txt1 . \$txt2</code>	Concatenation of <code>\$txt1</code> and <code>\$txt2</code>
.=	Concatena assign	<code>\$txt1 .= \$txt2</code>	Appends <code>\$txt2</code> to <code>\$txt1</code>

Operadors d'array

Els operadors d'array PHP s'utilitzen per comparar arrays.

Operator	Name	Example	Result
+	Union	<code>\$x + \$y</code>	Union of <code>\$x</code> and <code>\$y</code>
==	Equality	<code>\$x == \$y</code>	True if <code>\$x</code> and <code>\$y</code> have the same key/value pairs
===	Identity	<code>\$x === \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs in the same order and of the same types
!=	Inequality	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<>	Inequality	<code>\$x <> \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
!==	Non-identity	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not identical to <code>\$y</code>

Operadors d'assignació condicional

Els operadors d'assignació condicional PHP s'utilitzen per establir un valor en funció de les condicions:

Operator	Name	Example	Result
?:	Ternary	<code>\$x = expr1 ? expr2 : expr3</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr2</code> if <code>expr1</code> = TRUE. The value of <code>\$x</code> is <code>expr3</code> if <code>expr1</code> = FALSE
??	Null coalescing	<code>\$x = expr1 ?? expr2</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr1</code> if <code>expr1</code> exists, and is not NULL. If <code>expr1</code> does not exist, or is NULL, the value of <code>\$x</code> is <code>expr2</code> . Introduced in PHP 7

9. if..else

Molt sovint, quan escriviu codi, voleu realitzar diferents accions diferents en funció d'una condició. Podeu utilitzar declaracions condicionals al vostre codi per fer-ho.

A PHP tenim les següents declaracions condicionals:

if declaració: executa algun codi si una condició és certa.

if...else declaració: executa algun codi si una condició és certa i un altre codi si aquesta condició és falsa

if...elseif...else declaració: executa codis diferents per a més de dues condicions

La declaració if

La instrucció if executa algun codi si una condició és certa.

```
if (condition) {  
    // code to be executed if condition is true;  
}
```

Exemple

```
if (5 > 3) {  
    echo "Have a good day!";  
}
```

També podem utilitzar variables en el if declaració:

```
$t = 14;  
  
if ($t < 20) {  
    echo "Have a good day!";  
}
```

Operadors de comparació

If Les declaracions solen contenir condicions que comparen dos valors.

Comproveu si \$t és igual a 14:

```
$t = 14;  
  
if ($t == 14) {  
    echo "Have a good day!";  
}
```

Per comparar dos valors, hem d'utilitzar un operador de comparació.

Aquests són els operadors de comparació PHP per utilitzar-los if declaracions:

Operator	Name	Result
==	Equal	Returns true if the values are equal
===	Identical	Returns true if the values and data types are identical
!=	Not equal	Returns true if the values are not equal
<>	Not equal	Returns true if the values are not equal
!==	Not identical	Returns true if the values or data types are not identical
>	Greater than	Returns true if the first value is greater than the second value
<	Less than	Returns true if the first value is less than the second value
>=	>= to	Returns true if the first value is greater than, or equal to, the second value
<=	<= to	Returns true if the first value is less than, or equal to, the second value

Operadors lògics

Per comprovar més d'una condició, podem utilitzar operadors lògics, com ara && :

Comproveu si \$a és més gran que \$b, i si \$a és menys que \$c:

```
$a = 200;  
$b = 33;  
$c = 500;  
  
if ($a > $b && $a < $c ) {  
    echo "Both conditions are true";  
}
```

Aquests són els operadors lògics PHP per utilitzar en els if:

Operator	Name	Description
and	And	True if both conditions are true
&&	And	True if both conditions are true
or	Or	True if either condition is true
	Or	True if either condition is true
xor	Xor	True if either condition is true, but not both
!	Not	True if condition is not true

Podem comparar tantes condicions com vulguem en una declaració if:

Comproveu si \$a és 2, 3, 4, 5, 6 o 7:

```
$a = 5;  
  
if ($a == 2 || $a == 3 || $a == 4 || $a == 5 || $a == 6 || $a == 7) {  
    echo "$a is a number between 2 and 7";  
}
```

La declaració if...else

La instrucció if...else executa algun codi si una condició és certa i un altre codi si aquesta condició és falsa.

```
if (condition) {  
    // code to be executed if condition is true;  
} else {  
    // code to be executed if condition is false;  
}
```

Exemple

```
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
} else {  
    echo "Have a good night!";  
}
```

La declaració if...elseif...else

La instrucció if...elseif...else executa codis diferents per a més de dos condicions.

```
if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    // code to be executed if first condition is false and this condition is true;
} else {
    // code to be executed if all conditions are false;
}
```

Exemple

```
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
```

Acurta el if

Per escriure un codi més curt, podeu escriure la declaració if en una línia.

Exemple

```
$a = 5;

if ($a < 10) $b = "Hello";

echo $b
```

Exemple. Una línia if... else declaració:

```
$a = 13;

$b = $a < 10 ? "Hello" : "Good Bye";

echo $b;
```

Declaració if anidada

Pots tenir if declaracions dins if declaracions, això s'anomena declaracions d'if imbricades.

```
$a = 13;

if ($a > 10) {
    echo "Above 10";
    if ($a > 20) {
        echo " and also above 20";
    } else {
        echo " but not above 20";
    }
}
```

10. Bucles

Sovint, quan escriviu codi, voleu que el mateix bloc de codi s'execute de nou un cert nombre de vegades. Per tant, en lloc d'afegir diverses línies de codi pràcticament iguals en un script, nosaltres podem utilitzar bucles.

Els bucles s'utilitzen per executar el mateix bloc de codi una i altra vegada, sempre que una determinada condició siga certa.

while - Recorre un bloc de codi sempre que la condició especificada siga certa

do...while - fa un bucle a través d'un bloc de codi una vegada i després repeteix el bucle sempre que la condició especificada siga certa

for - Recorre un bloc de codi un nombre determinat de vegades

foreach - Recorre un bloc de codi per a cada element d'un array

while - Recorre un bloc de codi sempre que la condició especificada siga certa.

Imprimir \$i sempre que \$i és inferior a 6:

```
$i = 1;
while ($i < 6) {
    echo $i;
    $i++;
}
```

Nota: recordeu augmentar \$i, o si no, el bucle continuarà per sempre.

while no s'executa un nombre específic de vegades, però comprova després de cada iteració si la condició encara és certa.

La condició no ha de ser un comptador, sinó una operació o qualsevol condició que s'avalua com a vertader o fals.

Sintaxi alternativa

while també es pot escriure amb la declaració endwhile

Imprimir \$i sempre que \$i és inferior a 6:

```
$i = 1;
while ($i < 6):
    echo $i;
    $i++;
endwhile;
```

do...while – Executa el bloc de codi una vegada i després repeteix el bucle sempre que la condició siga certa.

Imprimir \$i sempre que \$i és inferior a 6:

```
$i = 1;

do {
    echo $i;
    $i++;
} while ($i < 6);
```

Nota: En el bucle do...while, la condició es comprova DESPRÉS d'executar-se. Això vol dir que el do...while s'executarà almenys una vegada, encara que la condició siga falsa. Vegeu l'exemple a continuació.

Vegem què passa si configurem el \$variable a 8 en lloc d'1, abans d'executar el mateix do...while bucle de nou:

```
$i = 8;

do {
    echo $i;
    $i++;
} while ($i < 6);
```

El codi s'executarà una vegada, encara que la condició mai siga certa.

for - Recorre un bloc de codi un nombre especificat de vegades. El for s'utilitza quan saps quantes vegades s'ha d'executar l'script.

```
for (expression1, expression2, expression3) {
    // code block
}
```

Imprimeix els números del 0 al 10:

```
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
```

El bucle foreach en arrays

L'ús més comú de la foreach, és per a recórrer els elements d'un array.

Recorre els elements d'un array indexat:

```
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
    echo "$x <br>";
}
```

Per a cada iteració de bucle, s'assigna el valor de l'element de l'array actual el variabe \$x. La iteració continua fins que arriba a l'últim element de l'array.

Claus i Valors

L'array de dalt és un array indexat , on el primer element té la clau 0, el segon té la clau 1, i així successivament.

Els arrays associatius són diferents, els arrays associatius utilitzen les claus amb nom que els hi assigneu i, quan passen per els arrays associatius, potser voldreu conservar la clau i el valor.

Això es pot fer especificant tant la clau com el valor al foreach:

Imprimeix tant la clau com el valor des de l'array \$members:


```
$members = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach ($members as $x => $y) {
    echo "$x : $y <br>";
}
```

El bucle foreach en objectes

El bucle foreach també es pot utilitzar per fer un bucle a través de les propietats d'un objecte:

Imprimeix els noms de propietat i els valors de la \$myCar objecte:

```
class Car {
    public $color;
    public $model;
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }
}

$myCar = new Car("red", "Volvo");

foreach ($myCar as $x => $y) {
    echo "$x: $y <br>";
}
```

Foreach Byref

Quan feu un bucle pels elements de l'array, qualsevol canvi fet a l'element de l'array, per defecte, NO afectarà l'array original:

Per defecte, canviar un element d'un array no afectarà l'array original:

```
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
    if ($x == "blue") $x = "pink";
}

var_dump($colors);
```

PERÒ, utilitzant el caràcter & en la declaració foreach, s'assigna l'element de l'array per referència, que es tradueix en que qualsevol canvi fet a l'element de l'array també es farà a l'array original:

En assignar els elements del array per referència, els canvis afectaran l'array original:

```
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as &$x) {
    if ($x == "blue") $x = "pink";
}

var_dump($colors);
```

Sintaxi alternativa

La sintaxi foreach de bucle també es pot escriure amb el endforeach declaració com aquesta

Recorre els elements d'un array indexat:

```
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $x) :  
    echo "$x <br>";  
endforeach;
```

11. Funcions

El poder real de PHP prové de les seves funcions. PHP té més de 1000 funcions integrades que es poden cridar directament des d'un programa, per interpretar una tasca concreta. A més, podeu crear les vostres propies funcions personalitzades.

Funcions definides per l'usuari de PHP

A més de les funcions PHP incorporades, és possible crear les vostres pròpies funcions.

Una funció és un bloc de sentències que es poden utilitzar repetidament en un programa. Una funció no s'executarà automàticament quan es carrega una pàgina. Una funció s'executarà mitjançant una crida a la funció.

Crea una funció

Una declaració de funció definida per l'usuari comença amb la paraula clau function, seguit del nom de la funció:

```
function myMessage() {  
    echo "Hello world!";  
}
```

Nota: el nom d'una funció ha de començar amb una lletra o un guió baix. Els noms de les funcions NO distingeixen entre majúscules i minúscules.

Crida una funció

Per a cridar a la funció, només cal escriure el seu nom seguit de parèntesis ():

```
function myMessage() {  
    echo "Hello world!";  
}  
  
myMessage();
```

La funció mostra "Hola món!".

Arguments de la funció PHP

La informació es pot passar a les funcions mitjançant arguments. Un argument és just com una variable.

Els arguments s'especifiquen després del nom de la funció, dins dels parèntesis. Podeu afegir tants arguments com vulgueu, només heu de separar-los amb una coma.

L'exemple següent té una funció amb un argument (\$fname). FamilyName() és el nom de la funció.

```
function familyName($fname) {  
    echo "$fname Refsnes.<br>";  
}  
  
familyName("Jani");  
familyName("Hege");  
familyName("Stale");  
familyName("Kai Jim");  
familyName("Borge");
```

L'exemple següent té una funció amb dos arguments (\$fname, \$year):

```
function familyName($fname, $year) {  
    echo "$fname Refsnes. Born in $year <br>";  
}  
  
familyName("Hege", "1975");  
familyName("Stale", "1978");  
familyName("Kai Jim", "1983");
```

Valors de retorn

Per permetre que una funció torne un valor, utilitzeu la declaració return:

```
function sum($x, $y) {  
    $z = $x + $y;  
    return $z;  
}  
  
echo "5 + 10 = " . sum(5, 10) . "<br>";  
echo "7 + 13 = " . sum(7, 13) . "<br>";  
echo "2 + 4 = " . sum(2, 4);
```

Passant arguments per referència

En PHP, els arguments es passen normalment per valor, el que significa que una còpia del valor ho és utilitzat a la funció i la variable que s'ha passat a la funció no es pot canviar.

Quan es passa un argument de funció per referència, els canvis a l'argument també canvien la variable que s'ha passat. Per convertir un argument de funció en una referència s'utilitza l'operador &:

```
function add_five(&$value) {  
    $value += 5;  
}  
  
$num = 2;  
add_five($num);  
echo $num;
```

Nombre variable d'arguments

Mitjançant l'ús de l'operador ... davant del paràmetre de funció, la funció accepta un nombre desconegut d'arguments. Això també s'anomena funció variàdica.

L'argument de la funció variàdica es converteix en un array.

Una funció que no sap quants arguments obtindrà:

```
function sumMyNumbers(...$x) {
    $n = 0;
    $len = count($x);
    for($i = 0; $i < $len; $i++) {
        $n += $x[$i];
    }
    return $n;
}

$a = sumMyNumbers(5, 2, 6, 2, 7, 7);
echo $a;
```

Només podeu tenir un argument amb longitud variable, i ha de ser l'últim argument.

L'argument variàdic ha de ser l'últim argument:

```
function myFamily($lastname, ...$firstname) {
    $txt = "";
    $len = count($firstname);
    for($i = 0; $i < $len; $i++) {
        $txt = $txt."Hi, $firstname[$i] $lastname.<br>";
    }
    return $txt;
}

$a = myFamily("Doe", "Jane", "John", "Joey");
echo $a;
```

PHP és un llenguatge poc tipat

En els exemples anteriors, observem que no havíem de dir en PHP de quin tipus de dades és la variable.

PHP associa automàticament un tipus de dades a la variable, en funció del seu valor. Com que els tipus de dades no s'estableixen en un sentit estricte, podeu fer coses com ara afegint una cadena a un nombre enter sense provocar un error.

A PHP 7, es van afegir declaracions de tipus. Això ens dona una opció per especificar el tipus de dades esperat quan es declara una funció i afegint el strict, llançarà un "Error fatal" si el tipus de dades no coincideix.

En l'exemple següent, intentem enviar tant un número com una cadena a funcionar sense utilitzar strict:

```
function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return
10
```

Per concretar strict hem d'establir `declare(strict_types=1);`. Això ha d'estar a la primera línia del fitxer PHP.

En l'exemple següent, intentem enviar tant un número com una cadena a funció, però ací hem afegit la funció strict declaració:

```
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be
thrown
?>
```

PHP 7 també admet declaracions de tipus per a return. Igual que amb la declaració de tipus per als arguments de funció, en activar el requisit estricte, llançarà un "Fatal Error" en una discrepància de tipus.

Per declarar un tipus per a la funció retornada, afegiu dos punts (:) i el tipus just abans de l'obertura de la clau { quan es declara la funció.

A l'exemple següent especifiquem el tipus de retorn per a la funció:

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
    return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

12. Arrays

Un array emmagatzema diversos valors en una sola variable:

```
$cars = array("Volvo", "BMW", "Toyota");
```

Què és un Array?

Un array és una variable especial que pot contenir molts valors en un sol nom, i podeu accedir als valors fent referència a un número com a índex.

Tipus d'arrays

A PHP, hi ha tres tipus d'arrays:

Arrays indexades : arrays amb un índex numèric

Arrays associatius : arrays amb claus de nom

Arrays multidimensionals : arrays que contenen un o més arrays

Treballant amb arrays

En aquest tutorial aprendràs a treballar amb arrays, com ara:

- Crear array

- accés a arrays
- Actualitza arrays
- Afegeix elements un array
- Elimina elements d'un array
- Ordena array

Elements d'un array

Els elements d'un array poden ser de qualsevol tipus de dades.

Els més habituals són les cadenes i els números (int, float), però els elements d'array també poden ser objectes, funcions o fins i tot arrays.

Podeu tenir diferents tipus de dades al mateix array.

Elements d'array de quatre tipus de dades diferents:

```
$myArr = array("Volvo", 15, ["apples", "bananas"], myFunction);
```

Funcions d'array

La força real dels arrays en PHP són les funcions d'array integrades, com ara count() per a comptar elements d'un array:

Quants elements hi ha al array \$cars:

```
$cars = array("Volvo", "BMW", "Toyota");  
echo count($cars);
```

Arrays indexats

Als arrays indexats cada element té un número d'índex.

Per defecte, el primer element té l'índex 0, el segon element té l'element 1, etc.

Creeu i visualitzeu un array indexat:

```
$cars = array("Volvo", "BMW", "Toyota");  
var_dump($cars);
```

Accés a arrays indexats

Per accedir a un element d'un array, podeu fer ho amb el número d'índex.

Mostra el primer element de l'array:

```
$cars = array("Volvo", "BMW", "Toyota");  
echo $cars[0];
```

Canvia el valor

Per canviar el valor d'un element d'array, utilitzeu el número d'índex:

Canviar el valor del segon element:

```
$cars = array("Volvo", "BMW", "Toyota");  
$cars[1] = "Ford";  
var_dump($cars);
```

Recorrer un array indexat

Per fer un bucle i imprimir tots els valors d'un array indexat, podeu utilitzar el bucle foreach:

Mostra tots els elements d'un array:

```
$cars = array("Volvo", "BMW", "Toyota");  
foreach ($cars as $x) {  
    echo "$x <br>";  
}
```

La clau d'un array indexat és un número, per defecte el primer element és 0 i el segon és 1, etc., però hi ha excepcions.

Els elements nous obtenen el següent número d'índex, és a dir, un més alt que l'índex existent més alt.

Per tant, si teniu un array com aquest:

```
$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";
```

I si feu servir array_push() per afegir un element nou, el nou element obtindrà l'índex 3:

```
array_push($cars, "Ford");  
var_dump($cars);
```

Però si teniu un array amb números d'índex aleatoris, com aquest:

```
$cars[5] = "Volvo";  
$cars[7] = "BMW";  
$cars[14] = "Toyota";
```

I si feu servir array_push() per a afegir un element nou, quin serà el número índex del nou element?

```
array_push($cars, "Ford");  
var_dump($cars);
```

Arrays associatius

Els arrays associatius són arrays que utilitzen claus de nom que se li assignen.

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);  
var_dump($car);
```

Accés a arrays associatius

Per accedir a un element d'array, podeu fer referència al nom de la clau.
e

Mostra el model del cotxe:

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);  
echo $car["model"];
```

Canvia el valor

Per canviar el valor d'un element d'array, utilitzeu el nom de la clau:

Canvia el year:

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);  
$car["year"] = 2024;  
var_dump($car);
```

Bucle a través d'un array associatiu

Per fer un bucle i imprimir tots els valors d'un array associatiu, podeu utilitzar el foreach, així:

Mostra tots els elements, claus i valors d'un array:

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);  
  
foreach ($car as $x => $y) {  
    echo "$x: $y <br>";  
}
```

Crea un array

Podeu crear arrays utilitzant la funció array():

```
$cars = array("Volvo", "BMW", "Toyota");
```

També podeu utilitzar una sintaxi més curta utilitzant el [] claudàtors:

```
$cars = ["Volvo", "BMW", "Toyota"];
```

Múltiples línies

Els salts de línia no són importants, de manera que una declaració d'array pot abastar diverses línies:

```
$cars = [  
    "Volvo",  
    "BMW",  
    "Toyota"  
];
```

Coma posterior

Es permet una coma després de l'últim element:

```
$cars = [  
    "Volvo",  
    "BMW",  
    "Toyota",  
];
```



```
];
```

Claus d'array

Quan es creen arrays indexats, les claus es donen automàticament, a partir de 0 i augmentades en 1 per cada element, de manera que l'array anterior també es podria crear amb claus:

```
$cars = [  
    0 => "Volvo",  
    1 => "BMW",  
    2 => "Toyota"  
];
```

Declara el array buit

Primer podeu declarar un array buit i afegir-hi elements més tard:

```
$cars = [];  
$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";
```

Barreja de claus d'array

Podeu tenir arrays amb claus indexades i amb nom:

```
$myArr = [];  
$myArr[0] = "apples";  
$myArr[1] = "bananas";  
$myArr["fruit"] = "cherries";
```

Accés a l'element de l'array

Per accedir a un element de l'array, podeu consultar el número d'índex dels arrays indexats, i el nom de la clau per als arrays associatius.

Accediu a un element fent referència al seu número d'índex:

```
$cars = array("Volvo", "BMW", "Toyota");  
echo $cars[2];
```

Per accedir a elements d'un array associatiu, utilitzeu el nom de la clau:

Accediu a un element fent referència al seu nom clau:

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);  
echo $cars["year"];
```

Cometes dobles o simples

Podeu utilitzar cometes simples i dobles quan accediu a un array:

```
echo $cars["model"];  
echo $cars['model'];
```

Executar un element de funció

Els elements d'un array poden ser de qualsevol tipus de dades, inclosa la funció.

Per executar aquesta funció, utilitzeu el número d'índex seguit de parèntesis ():

Executeu un element de funció:

```
function myFunction() {  
    echo "I come from a function!";  
}  
  
$myArr = array("Volvo", 15, myFunction);  
  
$myArr[2]();
```

Utilitzeu el nom de la clau quan la funció siga un element d'un array associatiu:

Executeu la funció fent referència al nom de la clau:

```
function myFunction() {  
    echo "I come from a function!";  
}  
  
$myArr = array("car" => "Volvo", "age" => 15, "message" => myFunction);  
  
$myArr["message"]();
```

Bucle a través d'un array associatiu

Per a fer un bucle i imprimir tots els valors d'un array associatiu, podeu utilitzar el foreach, així:

Mostra tots els elements, claus i valors de l'array:

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);  
  
foreach ($car as $x => $y) {  
    echo "$x: $y <br>";  
}
```

Recorre un array indexat

Per fer un bucle i imprimir tots els valors d'un array indexat, podeu utilitzar el foreach:

Mostra tots els elements de l'array:

```
$cars = array("Volvo", "BMW", "Toyota");  
foreach ($cars as $x) {  
    echo "$x <br>";  
}
```

Actualitza l'element del array

Per actualitzar un element d'un array existent, podeu consultar el número d'índex de l'array indexat, i el nom de la clau per a els arrays associatius.

Canvieu el segon element de l'array de "BMW" a "Ford":

```
$cars = array("Volvo", "BMW", "Toyota");  
$cars[1] = "Ford";
```

Per a actualitzar elements d'un array associatiu , utilitzeu el nom de la clau:

Actualització de l'any fins al 2024:

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);  
$cars["year"] = 2024;
```

Actualitza els elements de l'array amb un bucle Foreach

Hi ha diferents tècniques quan es canvien els valors de l'element amb foreach.

Una manera és inserir el caràcter & quan assignem el valor de l'element per referència i, per tant, assegurar-se que qualsevol canvi fet amb l'element de l'array dins del bucle es farà a l'array original:

Canvieu TOTS els valors dels elements a "Ford":

```
$cars = array("Volvo", "BMW", "Toyota");  
foreach ($cars as &$x) {  
    $x = "Ford";  
}  
unset($x);  
var_dump($cars);
```

Sense el unset(\$x), la variable \$x romandrà com a referència a l'últim element de l'array.

Afegeix un element a l'array

Per afegir elements a un array existent, podeu utilitzar el claudàtor [] sintaxi.

Afegiu un element més al array fruits:

```
$fruits = array("Apple", "Banana", "Cherry");  
$fruits[] = "Orange";
```

Arrays associatius

Per afegir elements a un array associatiu o a un array de claus/valors, feu servir claudàtors [] per a la clau i assigneu un valor amb el =.

Afegiu un element al array car:

```
$cars = array("brand" => "Ford", "model" => "Mustang");  
$cars["color"] = "Red";
```

Afegiu diversos elements d'array

Per afegir diversos elements a un array existent, utilitzeu array_push().

Afegiu tres elements al array fruits:

```
$fruits = array("Apple", "Banana", "Cherry");  
array_push($fruits, "Orange", "Kiwi", "Lemon");
```

Afegiu diversos elements als vectors associatius

Per afegir diversos elements a un vector existent, podeu utilitzar el += operador.

Afegiu dos elements al array cars:

```
$cars = array("brand" => "Ford", "model" => "Mustang");  
$cars += ["color" => "red", "year" => 1964];
```

Elimina un element d'un array

Per eliminar un element existent d'unarray, podeu utilitzar el array_splice().

Amb el array_splice() especifiqueu l'índex (on començar) i quants elements voleu suprimir.

Elimina el segon element:

```
$cars = array("Volvo", "BMW", "Toyota");  
array_splice($cars, 1, 1);
```

Després de la supressió, l'array es reindexa automàticament, començant a l'índex 0.

També podeu utilitzar el unset() funció per eliminar els elements de l'array existents.

Nota: El unset() no reorganitza els índexs, significa que després de la supressió, l'array ja no contindrà els índexs que falten.

Elimina el segon element:

```
$cars = array("Volvo", "BMW", "Toyota");  
unset($cars[1]);
```

Elimina diversos elements de l'array

Per eliminar diversos elements, el array_splice() pren un paràmetre de longitud que us permet especificar el nombre d'elements a eliminar.

Traieu 2 elements, començant pel segon element (índex 1):

```
$cars = array("Volvo", "BMW", "Toyota");  
array_splice($cars, 1, 2);
```

Elimina l'element d'un array associatiu

Per eliminar elements d'un array associatiu, podeu utilitzar el unset().

Especifiqueu la clau de l'element que voleu suprimir.

Elimina el "model":

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);  
unset($cars["model"]);
```

Utilitzant la funció array_diff

També podeu utilitzar el array_diff() per eliminar elements d'un array associatiu.

Aquesta funció retorna un array nou, sense els elements especificats.

Creeu un array nou, sense "Mustang" i "1964":

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);  
$newarray = array_diff($cars, ["Mustang", 1964]);
```

El array_pop() elimina l'últim element d'un array.

Elimina l'últim element:

```
$cars = array("Volvo", "BMW", "Toyota");  
array_pop($cars);
```

El array_shift() elimina el primer element d'un array.

Elimina el primer element:

```
$cars = array("Volvo", "BMW", "Toyota");  
array_shift($cars);
```

Funcions d'ordenació per a arrays

Ara vorem les següents funcions d'ordenació d'arrays PHP:

- sort() - Ordena els arrays en ordre ascendent
- rsort() - Ordena els arrays en ordre descendent
- asort() - ordena els arrays associatius en ordre ascendent, segons el valor
- ksort() - ordena els arrays associatius en ordre ascendent, segons la clau
- arsort() - ordena els arrays associatius en ordre descendent, segons el valor
- krsort() - ordena els arrays associatius en ordre descendent, segons la clau

Ordena l'array en ordre ascendent - sort()

L'exemple següent ordena els elements de l'array \$cars en ordre alfabètic ascendent:

```
$cars = array("Volvo", "BMW", "Toyota");  
sort($cars);
```

Ara els nombres:

```
$numbers = array(4, 6, 2, 22, 11);  
sort($numbers);
```

Ordena l'array (ordre ascendent), segons el valor - asort()

L'exemple següent ordena l'array associatiu en ordre ascendent, segons el valor:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
asort($age);
```

Ordena l'array (ordre ascendent), segons la clau - ksort()

L'exemple següent ordena un array associatiu en ordre ascendent, segons la clau:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
ksort($age);
```

Arrays multidimensionals

Un array multidimensional és un array que conté un o més arrays.

PHP admet arrays multidimensionals de dos, tres, quatre, cinc o més nivells de profunditat.

La dimensió d'un array indica el nombre d'índexs que necessiteu per seleccionar un element.

Arrays bidimensionals

Un array bidimensional és un array de arrays

Primer, mireu la taula següent:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

Podem emmagatzemar les dades de la taula anterior en un array bidimensional, com aquest:

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

Ara l'array bidimensional \$cars conté quatre array i té dos índexs: fila i columna.

Per tenir accés als elements de l'array \$cars hem d'apuntar als índexs (fila i columna):

```
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."<br>;  
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."<br>;  
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."<br>;  
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."<br>;
```

També podem posar a for bucle dins d'un altre for per obtenir els elements de l'array \$cars (encara hem d'apuntar als dos índexs):

```
for ($row = 0; $row < 4; $row++) {  
    echo "<p><b>Row number $row</b></p>";  
    echo "<ul>";  
    for ($col = 0; $col < 3; $col++) {
```

```

        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}

```

Totes les funcions d'array en PHP

PHP té un conjunt de funcions integrades que podeu utilitzar en arrays.

Function	Description
array()	Creates an array
array_change_key_case()	Changes all keys in an array to lowercase or uppercase
array_chunk()	Splits an array into chunks of arrays
array_column()	Returns the values from a single column in the input array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays, and returns the differences (compare values only)
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_diff_uassoc()	Compare arrays, and returns the differences (compare keys and values, using a user-defined key comparison function)
array_diff_ukey()	Compare arrays, and returns the differences (compare keys only, using a user-defined key comparison function)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the values of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays, and returns the matches (compare values only)
array_intersect_assoc()	Compare arrays and returns the matches (compare keys and values)
array_intersect_key()	Compare arrays, and returns the matches (compare keys only)
array_intersect_uassoc()	Compare arrays, and returns the matches (compare keys and values, using a user-defined key comparison function)
array_intersect_ukey()	Compare arrays, and returns the matches (compare keys only, using a user-defined key comparison function)
array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_merge_recursive()	Merges one or more arrays into one array recursively
array_multisort()	Sorts multiple or multi-dimensional arrays
array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user-defined function

<code>array_replace()</code>	Replaces the values of the first array with the values from following arrays
<code>array_replace_recursive()</code>	Replaces the values of the first array with the values from following arrays recursively
<code>array_reverse()</code>	Returns an array in the reverse order
<code>array_search()</code>	Searches an array for a given value and returns the key
<code>array_shift()</code>	Removes the first element from an array, and returns the value of the removed element
<code>array_slice()</code>	Returns selected parts of an array
<code>array_splice()</code>	Removes and replaces specified elements of an array
<code>array_sum()</code>	Returns the sum of the values in an array
<code>array_udiff()</code>	Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function)
<code>array_udiff_assoc()</code>	Compare arrays, and returns the differences (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)
<code>array_udiff_uassoc()</code>	Compare arrays, and returns the differences (compare keys and values, using two user-defined key comparison functions)
<code>array_uintersect()</code>	Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function)
<code>array_uintersect_assoc()</code>	Compare arrays, and returns the matches (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)
<code>array_uintersect_uassoc()</code>	Compare arrays, and returns the matches (compare keys and values, using two user-defined key comparison functions)
<code>array_unique()</code>	Removes duplicate values from an array
<code>array_unshift()</code>	Adds one or more elements to the beginning of an array
<code>array_values()</code>	Returns all the values of an array
<code>array_walk()</code>	Applies a user function to every member of an array
<code>array_walk_recursive()</code>	Applies a user function recursively to every member of an array
<code>arsort()</code>	Sorts an associative array in descending order, according to the value
<code>asort()</code>	Sorts an associative array in ascending order, according to the value
<code>compact()</code>	Create array containing variables and their values
<code>count()</code>	Returns the number of elements in an array
<code>current()</code>	Returns the current element in an array
<code>each()</code>	Deprecated from PHP 7.2. Returns the current key and value pair from an array
<code>end()</code>	Sets the internal pointer of an array to its last element
<code>extract()</code>	Imports variables into the current symbol table from an array
<code>in_array()</code>	Checks if a specified value exists in an array
<code>key()</code>	Fetches a key from an array
<code>ksort()</code>	Sorts an associative array in descending order, according to the key
<code>ksort()</code>	Sorts an associative array in ascending order, according to the key
<code>list()</code>	Assigns variables as if they were an array
<code>natcasesort()</code>	Sorts an array using a case insensitive "natural order" algorithm
<code>natsort()</code>	Sorts an array using a "natural order" algorithm
<code>next()</code>	Advance the internal array pointer of an array
<code>pos()</code>	Alias of <code>current()</code>
<code>prev()</code>	Rewinds the internal array pointer
<code>range()</code>	Creates an array containing a range of elements

reset()	Sets the internal pointer of an array to its first element
rsort()	Sorts an indexed array in descending order
shuffle()	Shuffles an array
sizeof()	Alias of count()
sort()	Sorts an indexed array in ascending order
uasort()	Sorts an array by values using a user-defined comparison function and maintains the index association
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array by values using a user-defined comparison function

13. Superglobals

Algunes variables predefinides en PHP són "superglobals", el que significa que sempre són accessibles, independentment de l'abast, i podeu accedir-hi des de qualsevol funció, classe o fitxer sense haver de fer res especial.

Les variables superglobals de PHP són:

- \$GLOBALS
- \$_SERVER
- \$_REQUEST
- \$_POST
- \$_GET
- \$_FILES
- \$_ENV
- \$_COOKIE
- \$_SESSION

\$GLOBALS és un array que conté totes les variables globals. Les variables globals són variables a les quals es pot accedir des de qualsevol àmbit.

Les variables de l'abast més exterior són automàticament variables globals, i es poden utilitzar automàticament. per qualsevol àmbit, per exemple, dins d'una funció.

Per utilitzar una variable global dins d'una funció, has de definir-la com a global amb la paraula clau global, o referir-se a ella mitjançant l'ús de \$GLOBALS.

Referència a la variable global \$x dins d'una funció:

```
$x = 75;

function myfunction() {
    echo $GLOBALS['x'];
}

myfunction()
```

Això és diferent d'altres llenguatges de programació on les variables globals estan disponibles sense fer cap cosa específicament.

En PHP no obtens res (o un error) quan et refereixes a una variable global sense la sintaxi \$GLOBALS

També et pots referir a variables globals dins de les funcions definint-te a elles com a globals amb la paraula clau global.

Defineix \$x com a global dins d'una funció:

```
$x = 75;

function myfunction() {
    global $x;
    echo $x;
}

myfunction()
```

Crear Variables Globals

Les variables creades amb l'abast exterior són variables globals, ja siga si es creen utilitzant la sintaxi \$GLOBALS o no:

```
$x = 100;

echo $GLOBALS["x"];
echo $x;
```

Les variables creades dins d'una funció només pertanyen a aquesta funció, però es poden crear variables globals dins d'una funció mitjançant l'ús de la sintaxi \$GLOBALS:

Crear una variable global des de l'interior d'una funció, i utilitzar-la fora de la funció:

```
function myfunction() {
    $GLOBALS["x"] = 100;
}

myfunction();

echo $GLOBALS["x"];
echo $x;
```

\$_SERVER

\$_SERVER és una variable súper global de PHP que conté informació sobre les capçaleres, camins i ubicacions de l'script.

L'exemple següent mostra com utilitzar alguns dels elements \$_SERVER:

```
echo $_SERVER['PHP_SELF'];
echo $_SERVER['SERVER_NAME'];
echo $_SERVER['HTTP_HOST'];
echo $_SERVER['HTTP_REFERER'];
echo $_SERVER['HTTP_USER_AGENT'];
echo $_SERVER['SCRIPT_NAME'];
```

La taula següent enumera els elements més importants que conté \$_SERVER:

Element/Code	Description
\$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
\$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
\$_SERVER['SERVER_ADDR']	Returns the IP address of the host server

<code>\$_SERVER['SERVER_NAME']</code>	Returns the name of the host server (such as www.w3schools.com)
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string (such as Apache/2.2.24)
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as HTTP/1.1)
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (such as POST)
<code>\$_SERVER['REQUEST_TIME']</code>	Returns the timestamp of the start of the request (such as 1377687496)
<code>\$_SERVER['QUERY_STRING']</code>	Returns the query string if the page is accessed via a query string
<code>\$_SERVER['HTTP_ACCEPT']</code>	Returns the Accept header from the current request
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
<code>\$_SERVER['HTTP_HOST']</code>	Returns the Host header from the current request
<code>\$_SERVER['HTTP_REFERER']</code>	Returns the complete URL of the current page (not reliable because not all user-agents support it)
<code>\$_SERVER['HTTPS']</code>	Is the script queried through a secure HTTP protocol
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page
<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SERVER_ADMIN']</code>	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the web server for communication (such as 80)
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Returns the server version and virtual host name which are added to server-generated pages
<code>\$_SERVER['PATH_TRANSLATED']</code>	Returns the file system based path to the current script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	Returns the URI of the current page

\$_REQUEST

`$_REQUEST` és una variable súper global de PHP que conté les dades del formulari enviat i totes les dades de les galetes.

En altres paraules, `$_REQUEST` és un array que conté dades de `$_GET`, `$_POST`, i `$_COOKIE`.

Podeu accedir a aquestes dades amb la paraula clau `$_REQUEST` seguida del nom del camp del formulari o galeta, com aquesta:

```
$_REQUEST['firstname']
```

S'està fent servir `$_REQUEST` a les sol·licituds de `$_POST`. Les sol·licituds POST solen ser dades enviades des d'un formulari HTML.

Ací teniu un exemple de com podria ser un formulari HTML:

```
<html>
<body>

<form method="post" action="demo_request.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

</body>
```

```
</html>
```

Quan un usuari fa clic al botó d'enviament, les dades del formulari s'envien a un fitxer PHP especificat al atribut action de la etiqueta <form>.

Al fitxer d'acció podem utilitzar la variable \$_REQUEST per recollir el valor del camp d'entrada.

```
$name = $_REQUEST['fname'];  
echo $name;
```

A l'exemple següent hem posat el formulari HTML i el codi PHP al mateix fitxer PHP.

També hem afegit algunes línies addicionals per seguretat.

```
<html>  
<body>  
  
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
  
<?php  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
  $name = htmlspecialchars($_REQUEST['fname']);  
  if (empty($name)) {  
    echo "Name is empty";  
  } else {  
    echo $name;  
  }  
}  
?  
  
</body>  
</html>
```

\$_REQUEST en sol·licituds de \$_GET

La sol·licitud GET pot usar-se en enviaments de formularis com a l'exemple anterior, amb el atribut method de <form> establert a GET.

Les sol·licituds GET també poden ser dades d'una cadena de consulta (informació afegida després d'una adreça URL).

Ací teniu un exemple de com seria un hiperenllaç HTML, amb una cadena de consulta:

```
<html>  
<body>  
  
<a href="demo_phpfile.php?subject=PHP&web=w3schools.com">Test $GET</a>  
  
</body>  
</html>
```

Quan un usuari fa clic a l'enllaç, s'envien les dades de la cadena de consulta demo_phpfile.php.

Al fitxer PHP podem utilitzar la variable `$_REQUEST` per recollir el valor de la cadena de consulta.

El fitxer PHP `demo_phpfile.php`:

```
<html>
<body>

<?php
echo "Study " . $_REQUEST['subject'] . " at " . $_REQUEST['web'];
?>

</body>
</html>
```

`$_POST`

`$_POST` conté un array de variables rebudes mitjançant el mètode HTTP POST.

Hi ha dues maneres principals d'enviar variables mitjançant el mètode HTTP Post:

- Formularis HTML
- Sol·licituds HTTP de JavaScript

`$_POST` als formularis HTML

Un formulari HTML envia informació mitjançant el mètode HTTP POST si l'atribut `method` està establert a "POST".

Per demostrar-ho, comencem creant un formulari HTML senzill:

```
<html>
<body>

<form method="POST" action="demo_request.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

</body>
</html>
```

Quan un usuari fa clic al botó d'enviament, les dades del formulari s'envien a un fitxer PHP especificat al atribut `action` de la etiqueta `form`.

Al fitxer d'acció podem utilitzar la variable `$_POST` per a recollir el valor del camp d'entrada.

```
$name = $_POST['fname'];
echo $name;
```

A l'exemple següent hem posat el formulari HTML i el codi PHP al mateix fitxer PHP.

També hem afegit algunes línies addicionals per seguretat.

```
<html>
<body>

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

```

    Name: <input type="text" name="fname">
    <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = htmlspecialchars($_POST['fname']);
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>

```

\$_POST a les sol·licituds HTTP de JavaScript

Quan envieu una sol·licitud HTTP en JavaScript, podeu especificar que el mètode HTTP siga POST.

Per demostrar-ho comencem creant una funció JavaScript que conté una sol·licitud HTTP:

```

function myfunction() {
    const xhttp = new XMLHttpRequest();
    xhttp.open("POST", "demo_phpfile.php");
    xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhttp.onload = function() {
        document.getElementById("demo").innerHTML = this.responseText;
    }
    xhttp.send("fname=Mary");
}

```

El codi anterior farà:

- Inicia una sol·licitud HTTP
- Estableix el mètode HTTP a POST
- Estableix una capçalera de sol·licitud vàlida
- Creeu una funció per executar-la quan es faça la sol·licitud
- Envieu la sol·licitud HTTP amb una variable fname ajustat a Mary

Mireu la funció que s'executarà quan es faça la sol·licitud:

```

xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
}

```

Intentarà escriure una resposta de l'operació en un element HTML amb id="demo".

Fem una pàgina HTML amb aquest element, i també un botó que executela funció.

Si també afegim el JavaScript, la pàgina quedarà així:
Com publicar i rebre dades d'una sol·licitud HTTP:

```

<html>
<script>

```

```

function myfunction() {
    const xhttp = new XMLHttpRequest();
    xhttp.open("POST", "demo_ajax.php");
    xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhttp.onload = function() {
        document.getElementById("demo").innerHTML = this.responseText;
    }
    xhttp.send("fname=Mary");
}
}
</script>
<body>

<button onclick="myfunction()">Click me!</button>

<h1 id="demo"></h1>

</body>
</html>

```

Al fitxer PHP que rep aquesta sol·licitud HTTP (demo_ajax.php), simplement fem servir la variable \$_POST per a recuperar la variable fname i l'escriu com a resposta.

Fitxer PHP

```

$name = $_POST['fname'];
echo $name;
$_GET

```

\$_GET conté una matriu de variables rebudes mitjançant el mètode HTTP GET.

Hi ha dues maneres principals d'enviar variables mitjançant el mètode HTTP GET:

- Cadenes de consulta a l'URL
- Formularis HTML

Cadena de consulta a l'URL

Una cadena de consulta són dades afegides al final d'un URL. A l'enllaç, tot després del signe de ? de i forma part de la cadena de consulta:

```

<a href="demo_phpfile.php?subject=PHP&web=W3schools.com">Test $_GET</a>

```

La cadena de consulta anterior conté dos parells clau/valor:

```

subject=PHP
web=W3schools.com

```

Al fitxer PHP podem utilitzar la variable \$_GET per a recollir el valor de la cadena de consulta.

El fitxer PHP demo_phpfile.php:

```

<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

```

```
</body>
</html>
```

\$_GET en formularis HTML

Un formulari HTML envia informació mitjançant el mètode HTTP GET si l'atribut method del form està establert a "GET".

Per demostrar-ho, comencem creant un formulari HTML senzill:

Formulari HTML

```
<html>
<body>

<form action="welcome_get.php" method="GET">
  Name: <input type="text" name="name">
  E-mail: <input type="text" name="email">
  <input type="submit">
</form>

</body>
</html>
```

Quan un usuari fa clic al botó d'enviament, les dades del formulari s'envien a un fitxer PHP especificat a l'atribut action de la etiqueta <form>.

Els camps del formulari s'envien al fitxer PHP, amb la vostra entrada, com a cadenes de consulta:

```
welcome_get.php?name=John&email=john@example.com
```

Al fitxer d'acció podem utilitzar la variable \$_GET per a recollir el valor dels camps d'entrada.

Codi PHP dins de welcome_get.php pàgina:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

Penseu en SEURETAT quan processeu formularis PHP!

14. Expressions regulars

Una expressió regular és una seqüència de caràcters que forma un patró de cerca. Quan cerqueu dades en un text, podeu utilitzar aquest patró de cerca per descriure el que esteu buscant. Una expressió regular pot ser un sol caràcter o un patró més complicat.

Les expressions regulars es poden utilitzar per realitzar tot tipus de cerca de text i operacions de substitució de text.

En PHP, les expressions regulars són cadenes compostes per delimitadors, un patró i modificadors opcionals.

```
$exp = "/w3schools/i";
```

En l'exemple anterior, /és el delimitador , w3schools és el patró que s'està buscant, i i és un modificador que fa que la cerca no distingeix entre majúscules i minúscules.

El delimitador pot ser qualsevol caràcter que no siga una lletra, un número, una barra invertida o un espai. El delimitador més comú és la barra inclinada (/), però quan el vostre patró conté barres inclinades és convenient triar altres delimitadors com # o ~.

Funcions d'expressió regular

PHP ofereix una varietat de funcions que us permeten utilitzar expressions regulars. Les funcions més habituals són:

Function	Description
preg_match()	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string
Utilitzant preg_match()	

El preg_match()

La funció us dirà si una cadena conté coincidències d'un patró.

Utilitzeu una expressió regular per fer una cerca que no distingeix entre majúscules i minúscules de "w3schools" en una cadena:

```
$str = "Visit W3Schools";  
$pattern = "/w3schools/i";  
echo preg_match($pattern, $str);
```

Utilitzant preg_match_all()

El preg_match_all() us indicarà quantes coincidències s'han trobat un patró en un string.

Utilitzeu una expressió regular per fer un recompte que no distingeix entre majúscules i minúscules del nombre de aparicions de "ain" en una cadena:

```
$str = "The rain in SPAIN falls mainly on the plains.";  
$pattern = "/ain/i";  
echo preg_match_all($pattern, $str);
```

Utilitzant preg_replace()

El preg_replace() substituirà totes les coincidències del patró en una cadena per un altre string

Utilitzeu una expressió regular que no distingeix entre majúscules i minúscules per substituir Microsoft per W3Schools en una cadena:

```
$str = "Visit Microsoft!";  
$pattern = "/microsoft/i";
```

```
echo preg_replace($pattern, "w3Schools", $str);
```

Modificadors d'expressió regular

Els modificadors poden canviar com es realitza una cerca.

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for a match at the beginning or end of a string will now match the beginning or end of each line)
u	Enables correct matching of UTF-8 encoded patterns

Patrons d'expressió regular

Els claudàtors s'utilitzen per trobar una sèrie de caràcters:

Expression	Description
[abc]	Find one or many of the characters inside the brackets
[^abc]	Find any character NOT between the brackets
[a-z]	Find any character alphabetically between two letters
[A-z]	Find any character alphabetically between a specified upper-case letter and a specified lower-case letter
[A-Z]	Find any character alphabetically between two upper-case letters.
[123]	Find one or many of the digits inside the brackets
[0-5]	Find any digits between the two numbers
[0-9]	Find any digits

Metapersonatges

Els metacaràcters són personatges amb un significat especial:

Metacharacter	Description
	Find a match for any one of the patterns separated by as in: cat dog fish
.	Find any character
^	Finds a match as the beginning of a string as in: ^Hello
\$	Finds a match at the end of the string as in: World\$
\d	Find any digits
\D	Find any non-digits
\s	Find any whitespace character
\S	Find any non-whitespace character
\w	Find any alphabetical letter (a to Z) and digit (0 to 9)
\W	Find any non-alphabetical and non-digit character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantificadors

Els quantificadors defineixen quantitats:

Quantifier	Description
n+	Matches any string that contains at least one n

n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{3}	Matches any string that contains a sequence of 3 n's
n{2, 5}	Matches any string that contains a sequence of at least 2, but not more than 5 n's
n{3,}	Matches any string that contains a sequence of at least 3 n's

Nota: si la vostra expressió necessita cercar un dels caràcters especials, podeu utilitzar la barra invertida (\) per escapar-los. Per exemple, per cercar un o més signes d'interrogació podeu utilitzar la següent expressió: \$patró = '/\?+/';

Agrupació

Podeu utilitzar parèntesis () per aplicar quantificadors a patrons sencers. També es poden utilitzar per seleccionar parts del patró que s'utilitzaran com a coincidència.

Utilitzeu l'agrupació per cercar la paraula "plàtan" cercant ba seguida de dos casos de na :

```
$str = "Apples and bananas.";
$pattern = "/ba(na){2}/i";
echo preg_match($pattern, $str);
```

Tema 3: PHP Avançat



-
1. Formularis
 2. Data i hora
 3. Fitxers
 4. Cookies
 5. Sessions
 6. Filtres
 7. Funcions callback
 8. JSON
 9. Excepcions
-

1. Formularis

Els superglobals de PHP `$_GET` i `$_POST` s'utilitzen per recollir dades de formulari.

L'exemple següent mostra un formulari HTML senzill amb dos camps d'entrada i un botó d'enviament:

```
<html>
<body>

<form action="welcome.php" method="POST">
    Name: <input type="text" name="name"><br>
    E-mail: <input type="text" name="email"><br>
    <input type="submit">
</form>

</body>
</html>
```

Quan l'usuari omple el formulari anterior i fa clic al botó d'enviament, s'envien les dades del formulari per processar-lo a un fitxer PHP anomenat "welcome.php". Les dades del formulari s'envien amb el mètode HTTP POST.

Per mostrar les dades enviades, simplement es podeu fer ressò de totes les variables.

El "welcome.php" té aquest aspecte:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

El mateix resultat també es podria aconseguir mitjançant el mètode HTTP GET. El codi anterior és bastant senzill i no inclou cap validació.

GET vs. POST

Tant GET com POST creen una matriu (per exemple, `array(clau1 => valor1, clau2 => valor2, clau3 => valor3, ...)`). Aquesta matriu conté parells clau/valor, on les claus són els noms dels controls del formulari i els valors són les dades d'entrada de l'usuari.

Tant GET com POST es carreguen en `$_GET` i `$_POST`. Aquests són superglobals, el que significa que sempre són accessibles, independentment de l'abast, i que hi podeu accedir des de qualsevol funció, classe o fitxer sense haver de fer res especial.

`$_GET` és un array de variables passat a l'script actual mitjançant els paràmetres d'URL.

`$_POST` és un array de variables passat a l'script actual mitjançant el mètode HTTP POST.

Quan utilitzar GET?

La informació enviada des d'un formulari amb el mètode GET és visible per a tothom (tots els noms i els valors de les variables es mostren a l'URL). GET també té límits en la

quantitat de informació per a enviar. La limitació és d'uns 2000 caràcters. GET es pot utilitzar per enviar dades no sensibles.

Quan utilitzar POST?

La informació enviada des d'un formulari amb el mètode POST és invisible per als altres (tots els noms/valors estan incrustats dins del cos de la sol·licitud HTTP) i no té límits en la quantitat d'informació a enviar.

A més, POST admet funcionalitats avançades com ara suport per a diverses parts entrada binària durant la càrrega de fitxers al servidor.

Tanmateix, com que les variables no es mostren a l'URL, no és possible marcar la pàgina a les adreces d'interès.

Els desenvolupadors prefereixen POST per enviar dades de formularis.

Validació de formularis

Penseu en **SEGURETAT** quan processeu formularis PHP! Ara anem a mostrar com processar formularis PHP tenint en compte la seguretat. És important la validació adequada de les dades del formulari per protegir el vostre formulari dels pirates informàtics i els spammers!

El formulari HTML que treballarem en aquests punts contenen diversos camps d'entrada: camps de text obligatoris i opcionals, botons d'opció i un botó d'enviament:

Exemple de validació de formularis PHP

* Camp necessari

Nom: *

Correu electrònic: *

Lloc web:

Comentari:

Gènere: ☐ Dona ☐ Mascle ☐ Altres *

La teva entrada:

Les regles de validació de l'anterior formulari són les següents:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

Primer veurem el codi HTML senzill del formulari:

Camps de text: El nom, el correu electrònic i els camps del lloc web són elements d'entrada de text i el comentari el camp és una àrea de text.

El codi HTML té aquest aspecte:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Botons de ràdio: Els camps de gènere són botons d'opció i el codi HTML té aquest aspecte:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

El codi HTML del formulari té aquest aspecte:

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

Quan s'envia el formulari, les dades del formulari s'envien amb method="post".

Que es el \$_SERVER["PHP_SELF"] variable?

El \$_SERVER["PHP_SELF"] és una variable súper global que retorna el nom de fitxer que està executant l'script.

El \$_SERVER["PHP_SELF"] envia les dades del formulari enviat a la pròpia pàgina, en lloc de saltar a una pàgina diferent. D'aquesta manera, l'usuari rebrà missatges d'error a la mateixa pàgina que el formulari.

Que es el htmlspecialchars() funció?

La funció htmlspecialchars() converteix caràcters especials en entitats HTML. Això vol dir que substituirà caràcters HTML com < i > amb < i >. Això evita que els atacants exploten el codi injectant codi HTML o Javascript (Cross-site Scripting atacs) en formularis.

La variable \$_SERVER["PHP_SELF"] pot ser utilitzada pels pirates informàtics! Si s'utilitza PHP_SELF a la vostra pàgina, un usuari pot introduir una barra inclinada / i algunes ordres de Cross Site Scripting (XSS) per a ser executades.

Cross-site scripting (XSS) és un tipus de vulnerabilitat de seguretat informàtica es troba habitualment en aplicacions web. XSS permet als atacants injectar del costat del client script en pàgines web vistes per altres usuaris.

Suposem que tenim el següent formulari en una pàgina anomenada "test_form.php":

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

Ara, si un usuari introdueix l'URL normal a la barra d'adreces com "http://www.example.com/test_form.php", el codi anterior es traduirà a:

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

Fins ara, tot bé.

Tanmateix, tingueu en compte que un usuari introdueix l'URL següent a la barra d'adreces:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

En aquest cas, el codi anterior es traduirà a:

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

Aquest codi afegeix una etiqueta d'script i una ordre d'alerta. I quan es carrega la pàgina, el codi JavaScript s'executarà i l'usuari veurà un quadre d'alerta. Això és només un senzill exemple inofensiu de com es pot explotar la variable PHP_SELF.

Tingueu en compte que es pot afegir qualsevol codi JavaScript dins del fitxer etiqueta <script>!. Un pirata informàtic pot redirigir l'usuari a un fitxer d'un altre servidor, i aquest fitxer pot contenir codi maliciós que pot alterar les variables globals o enviar el formulari a un altre per desar les dades de l'usuari, per exemple.

Com evitar les explotacions de \$_SERVER["PHP_SELF"]?

Es poden evitar explotacions de \$_SERVER["PHP_SELF"] utilitzant la funció htmlspecialchars().

El codi del formulari hauria de ser així:

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

htmlspecialchars() converteix caràcters especials en entitats HTML. Ara, si l'usuari intenta explotar la variable PHP_SELF, donarà com a resultat la següent eixida:

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

L'intent d'explotació falla i no es fa cap mal!

Valideu les dades del formulari amb PHP

El primer que farem és passar totes les variables a través de la funció htmlspecialchars().

Quan fem servir el htmlspecialchars()funció; aleshores, si un usuari intenta enviar el següent en un camp de text:

```
<script>location.href('http://www.hacked.com')</script>
```

Això no s'executaria, perquè es desaria com a codi d'escapada HTML, com aquest:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

El codi ara és segur per mostrar-se en una pàgina o dins d'un correu electrònic.

També farem dues coses més quan l'usuari envie el formulari:

Elimina els caràcters innecessaris (espai addicional, tabulació, nova línia) de les dades d'entrada de l'usuari (amb la funció PHP trim())

Elimina les barres invertides \ de les dades d'entrada de l'usuari (amb la funció PHP stripslashes())

El següent pas és crear una funció que ens farà totes les comprovacions (cosa molt més convenient que escriure el mateix codi una i altra vegada).

Anomenarem la funció test_input().

Ara, podem comprovar cadascuna de les variables \$_POST amb la funció test_input(), i l'script té aquest aspecte:

```
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

Observeu que a l'inici de l'script, comprovem si el formulari ha estat enviat utilitzant \$_SERVER["REQUEST_METHOD"]. Si el REQUEST_METHOD és POST, doncs s'ha enviat el formulari i això hauria de ser validat. Si no s'ha enviat, salteu la validació i mostrar un formulari en blanc.

Tanmateix, a l'exemple anterior, tots els camps d'entrada són opcionals. El tot funciona bé encara que l'usuari no introdueix cap dada.

El següent pas és fer que els camps d'entrada siguin obligatoris i crear missatges d'error si és necessari.

Camps necessaris (requerits)

Del formulari de la imatge tenim que nom, correu i gènere són caps requerits. Aquests camps no poden estar buits i s'han d'omplir en el Formulari HTML.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)

Gender Required. Must select one

En el següent codi hem afegit algunes noves variables: \$nameErr, \$emailErr, \$genderErr, i \$websiteErr. . Aquestes variables d'error mantindran missatges d'error per a la Camps necessaris. També hem afegit una declaració if else per a cada variable \$_POST. Aquesta Comprovar si la variable \$_POST està buida (amb la funció PHP empty()) Si està buit, s'emmagatzema un missatge d'error en les diferents variables d'error. I si no està buit, envia les dades d'entrada de l'usuari a través de la funció test_input():

```
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

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

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = test_input($_POST["email"]);
    }

    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
    }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
```

Mostrar els missatges d'error

Després, en la forma HTML, afegim un text després de cada camp requerit, que genera el missatge d'error si és necessari (és a dir, si l'usuari intenta presentar el formulari sense omplir els camps requerits):

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

```
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
```

```

Website:

<?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
☐>* <?php echo $genderErr;?></span>
<br><br>


</form>

```

El següent pas és validar les dades d'entrada, és a dir "Fer que el camp del Nom continga només lletres i espai blanc, i el camp de correu electrònic continga un a l'adreça de correu electrònic vàlida, i, si s'omple, el camp del lloc web continga una URL vàlida.

Validar el nom

El codi següent mostra una manera senzilla de comprovar si el camp de nom només conté lletres, guionets, apòstrofs i espais blancs. Si el valor del camp de nom no és vàlid, emmagatzemarem un missatge d'error:

```

$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/", $name)) {
    $nameErr = "Only letters and white space allowed";
}

```

La funció `preg_match()` busca un string patró, tornant veritat si el patró existeix, i fals d'una altra manera.

Correu electrònic validat

La manera més fàcil i segura de comprovar si una adreça de correu electrònic està ben formada és utilitzar la funció `filter_var()`.

En el codi següent, si l'adreça de correu electrònic no està ben formada, s'emmagatzenarà un missatge d'error:

```

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

```

Validar una URL

El codi següent mostra una manera de comprovar si la sintaxi d'adreces URL és vàlida. Si la sintaxi de l'adreça URL no és vàlida, aleshores emmagatzenarem un missatge d'error:

```

$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?~_!:\.,;]*[-a-z0-9+&@#\/%~_]/i", $website)) {
    $websiteErr = "Invalid URL";
}

```

Ara tot: nom validat, correu electrònic i URL

Ara, el codi quedarà així:

```
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
        // check if name only contains letters and whitespace
        if (!preg_match("/^[a-zA-Z-' ]*$/", $name)) {
            $nameErr = "Only letters and white space allowed";
        }
    }

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = test_input($_POST["email"]);
        // check if e-mail address is well-formed
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
        }
    }

    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
        // check if URL address syntax is valid (this regular expression also allows
        // dashes in the URL)
        if (!preg_match("/\b(?:(:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?~_!|:.,;]*[-a-z0-9+&@#\/%~_!|:.,;]/i", $website)) {
            $websiteErr = "Invalid URL";
        }
    }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
```

Mantindre els valors del formulari després d'enviat

Per mostrar els valors en els camps d'entrada després que l'usuari preme el botó d'enviament, afegim un poc de codi.

A continuació, també hem de mostrar quin botó de ràdio es va marcar.

Ací tenim el codi complet:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = test_input($_POST["name"]);
        // check if name only contains letters and whitespace
        if (!preg_match("/^[a-zA-Z-' ]*$/", $name)) {
            $nameErr = "Only letters and white space allowed";
        }
    }

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = test_input($_POST["email"]);
        // check if e-mail address is well-formed
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
        }
    }

    if (empty($_POST["website"])) {
        $website = "";
    } else {
        $website = test_input($_POST["website"]);
        // check if URL address syntax is valid (this regular expression also allows
dashes in the URL)
        if (!preg_match("/\b(?:(:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?
=~|!|:,.;]*[-a-z0-9+&@#\/%=~|!|]/i", $website)) {
            $websiteErr = "Invalid URL";
        }
    }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else {
        $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else {
        $gender = test_input($_POST["gender"]);
    }
}
```

```

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    Name: <input type="text" name="name" value="<?php echo $name;?>">
    <span class="error">* <?php echo $nameErr;?></span>
    <br><br>
    E-mail: <input type="text" name="email" value="<?php echo $email;?>">
    <span class="error">* <?php echo $emailErr;?></span>
    <br><br>
    Website: <input type="text" name="website" value="<?php echo $website;?>">
    <span class="error"><?php echo $websiteErr;?></span>
    <br><br>
    Comment: <textarea name="comment" rows="5" cols="40"><?php echo
    $comment;?></textarea>
    <br><br>
    Gender:
    <input type="radio" name="gender" <?php if (isset($gender) &&
    $gender=="female") echo "checked";?> value="female">Female
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male")
    echo "checked";?> value="male">Male
    <input type="radio" name="gender" <?php if (isset($gender) &&
    $gender=="other") echo "checked";?> value="other">Other
    <span class="error">* <?php echo $genderErr;?></span>
    <br><br>
    <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>

```

2. Data i hora

La funció date() s'utilitza per a formatar una data i/o hora.

Sintaxi **date(format,timestamp)**

Parameter Description

format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

Obtenir la data

El paràmetre de format requerit de la funció `date()` especifica com es formatarà la data (o el temps).

Aquests són alguns dels caràcters que s'utilitzen habitualment per a dates:

- d - Representa el dia del mes (01 a 31)
- m - Representa un mes (01 a 12)
- Y - Representa un any (en quatre dígit)
- l ('L' minúscula) - Representa el dia de la setmana

Altres caràcters, com `"/"`, `" "`, o `"-"` també es poden inserir entre els Personatges per afegir format addicional.

Exemple de formats de la data d'huí de tres maneres diferents:

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

Utilitza la funció `date()` per a actualitzar automàticament l'any del copyright en el teu lloc web:

Obtenir el temps

Ací hi ha alguns caràcters que s'utilitzen habitualment per a l'hora:

- H - 24 hores de format d'una hora (00 a 23)
- h - format de 12 hores d'una hora amb zeros capdavanters (01 a 12)
- i - Minuts amb zeros de lideren (00 a 59)
- s - Seconds amb zeros de liderada (00 a 59)
- a - Inferior Ante Meridiem i Post meridem (am o pm)

L'exemple següent mostra l'hora en el format especificat:

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

Tingueu en compte que la funció `date()` retornarà la data/hora actual del servidor!

Obtenir la teua zona horària

Si l'hora que retornar el codi no és correcte, és probablement perquè el seu servidor està en un altre país o configurat per a una zona horària diferent.

Si necessites l'hora correcta en un determinat lloc, pots establir la zona horària que vulgues utilitzar.

L'exemple següent estableix la zona horària a "Amèrica/Nova_York", després mostra el temps actual en el format especificat:

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

Crear una data amb mktime()

El paràmetre opcional de segell de temps de la funció date() especifica un segell de temps. Si s'omet, s'utilitzarà la data i hora actual (com en els exemples de més amunt).

La funció mktime() retorna el segell de temps Unix per a una data. El segell Unix conté el nombre de segons entre l'Unix Epoch. (1 de gener de 1970 00:00:00 GMT) i el temps especificat.

Sintaxi: ***mktime(hour, minute, second, month, day, year)***

L'exemple següent crea una data i una hora amb la funció date() a partir d'una sèrie de paràmetres en la funció mktime()

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Crear una data a partir d'un string amb strtotime()

La funció strtotime() s'utilitza per convertir una cadena de data llegible per humans en una marca de temps Unix (el nombre de segons des de l'1 de gener de 1970 a les 00:00:00 GMT).

Sintaxi: ***strtotime(time, now)***

L'exemple següent crea una data i una hora a partir de la funció strtotime():

```
<?php
$d=strtotime("10:30pm April 15 2014");
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

PHP és bastant intel·ligent a l'hora de convertir una cadena en una data, de manera que podeu introduir diversos valors:

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

3. Fitxers

Inclou fitxers:

La instrucció include (o require) pren tot el text/codi/marcatge que existeix al fitxer especificat i el copia al fitxer que utilitza la instrucció include.

Incloure fitxers és molt útil quan voleu incloure el mateix PHP, HTML o text a diverses pàgines d'un lloc web.

És possible inserir el contingut d'un fitxer PHP en un altre fitxer PHP (abans que el servidor l'execute), amb la instrucció include o require.

Les declaracions include i require són idèntiques, excepte en cas d'errada:

- require produirà un error fatal (E_COMPILE_ERROR) i aturarà l'script
- include només produirà un avís (E_WARNING) i l'script continuarà

Per tant, si voleu que l'execució continue i mostre als usuaris l'eixida, encara que falte el fitxer include, utilitzeu la instrucció include. En cas contrari, en cas de codificació de FrameWork, CMS o d'una aplicació PHP complexa, utilitzeu sempre la instrucció require per incloure un fitxer important al flux d'execució. Això ajudarà a evitar comprometre la seguretat i la integritat de la vostra aplicació.

Incloure fitxers estalvia molta feina. Això vol dir que podeu crear una capçalera, un peu de pàgina o un fitxer de menú estàndard per a totes les vostres pàgines web. Aleshores, quan cal actualitzar la capçalera, només heu d'actualitzar el fitxer d'inclusió de la capçalera.

Sintaxi: ***include 'filename';*** or ***require 'filename';***

Exemple 1

Suposem que tenim un fitxer de peu de pàgina estàndard anomenat "footer.php", que té aquest aspecte:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

Per incloure el fitxer de peu de pàgina en una pàgina, utilitzeu la instrucció include:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

Exemple 2

Suposem que tenim un fitxer de menú estàndard anomenat "menu.php":

```
<?php
echo '<a href="/default.asp">Home</a> -
<a href="/html/default.asp">HTML Tutorial</a> -
```

```
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>;
?>
```

Totes les pàgines del lloc web haurien d'utilitzar aquest fitxer de menú. A continuació es mostra com es pot fer (estem utilitzant un element <div> perquè el menú es puga dissenyar fàcilment amb CSS més endavant):

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

Exemple 3

Suposem que tenim un fitxer anomenat "vars.php", amb algunes variables definides:

```
<?php
$color='red';
$car='BMW';
?>
```

Aleshores, si incloem el fitxer "vars.php", les variables es poden utilitzar al fitxer el crida:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php include 'vars.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

Incloure vs. requerir

La declaració required també s'utilitza per incloure un fitxer al codi PHP.

Podem dir que hi ha una gran diferència entre incloure i requerir. Quan s'inclou un fitxer amb la instrucció include i PHP no el troba, l'script continuarà executant-se.

Si fem el mateix amb la instrucció require, l'execució de l'script mor després que la instrucció require i retorna un error fatal.

Utilitzeu require quan l'aplicació requereix el fitxer. Utilitzeu include quan el fitxer no siga necessari i l'aplicació pot continuar encara que no trobe el fitxer.

Manipulant fitxers

PHP té diverses funcions per a crear, llegir, carregar i editar fitxers.

Aneu amb compte quan manipuleu fitxers! Quan manipuleu fitxers heu de tenir molta cura. Podeu fer molt de mal si feu alguna cosa malament. Els errors habituals són: editar el fitxer incorrecte, omplir un disc dur amb dades d'escombraries i suprimir el contingut d'un fitxer per accident.

Funció PHP `readfile()`

La funció `readfile()` llegeix un fitxer i l'escriu a la memòria intermèdia d'eixida.

Suposem que tenim un fitxer de text anomenat "webdictionary.txt", emmagatzemat al servidor, que té aquest aspecte:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXTensible Markup Language
```

El codi PHP per llegir el fitxer i escriure-lo a la memòria intermèdia d'eixida és el següent (la funció `readfile()` retorna el nombre de bytes llegits en cas d'èxit):

```
<!DOCTYPE html>
<html>
<body>

<?php
echo readfile("webdictionary.txt");
?>

</body>
</html>
```

Obrir fitxer - `fopen()`

El millor mètode per a obrir fitxers és amb la funció `fopen()`. Aquesta funció us ofereix més opcions que la funció `readfile()`.

Utilitzarem el fitxer de text "webdictionary.txt" com a exemple:

El primer paràmetre de `fopen()` conté el nom del fitxer que s'ha d'obrir i el segon paràmetre especifica en quin mode s'ha d'obrir el fitxer. L'exemple següent també genera un missatge si la funció `fopen()` no pot obrir el fitxer especificat:

```
<!DOCTYPE html>
<html>
<body>
    <?php
        $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
        echo fread($myfile, filesize("webdictionary.txt"));
        fclose($myfile);
    ?>
</body>
</html>
```

El fitxer es pot obrir en un dels modes següents:

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

Lectura de fitxer - fread()

La funció fread() llegeix des d'un fitxer obert.

El primer paràmetre de fread() conté el nom del fitxer per a llegir i el segon paràmetre especifica el nombre màxim de bytes per llegir.

El següent codi PHP llegeix el fitxer "webdictionary.txt" fins al final:

```
fread($myfile, filesize("webdictionary.txt"));
```

Tanca el fitxer - fclose()

La funció fclose() s'utilitza per a tancar un fitxer obert. És una bona pràctica de programació tancar tots els fitxers un cop s'haja acabat amb ells. No voldreu que un fitxer obert funcione al vostre servidor ocupant recursos!

Requereix la funció fclose() el nom del fitxer (o una variable que conté el nom del fitxer) que volem tancar:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

Llegir una línia d'un fitxer - fgets()

La fgets() funció s'utilitza per llegir una única línia d'un fitxer. L'exemple següent mostra la primera línia del fitxer "webdictionary.txt":

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Nota: després d'una crida a la funció `fgets()`, el punter del fitxer s'ha mogut a la línia següent.

Comprova el final de fitxer - `feof()`

La funció `feof()` comprova si s'ha arribat al "final del fitxer" (EOF).

La funció `feof()` és útil per fer un bucle a través de dades de longitud desconeguda.

L'exemple següent llegeix el fitxer "webdictionary.txt" línia per línia, fins que s'arriba al final del fitxer:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
    echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

Llegir un sol caràcter - `fgetc()`

La funció `fgetc()` s'utilitza per llegir un sol caràcter d'un fitxer.

L'exemple següent llegeix el fitxer "webdictionary.txt" caràcter per caràcter, fins que s'arriba al final del fitxer:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
    echo fgetc($myfile);
}
fclose($myfile);
?>
```

Nota: després d'una crida a la funció `fgetc()`, el punter del fitxer es mou al caràcter següent.

Crea un fitxer - `fopen()`

La funció `fopen()` també s'utilitza per crear un fitxer. Potser açò resulta una mica confús, però a PHP, un fitxer es crea amb la mateixa funció que s'utilitza per a obrir fitxers.

Si utilitzeu `fopen()` cridant a un fitxer que no existeix, el crearà.

L'exemple següent crea un fitxer nou anomenat "testfile.txt". El fitxer es crearà al mateix directori on resideix el codi PHP:

```
$myfile = fopen("testfile.txt", "w")
```

Permisos de fitxers PHP

Si teniu errors quan intenteu executar el codi anterior, comproveu que heu concedit accés al vostre fitxer PHP per escriure informació al disc dur.

Esriptura al fitxer - fwrite()

La funció fwrite() s'utilitza per a escriure en un fitxer.

El primer paràmetre de fwrite() conté el nom del fitxer a escriure i el segon paràmetre és la cadena que s'ha d'escriure.

L'exemple següent escriu un parell de noms en un fitxer nou anomenat "newfile.txt":

```
<?php
    $myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
    $txt = "John Doe\n";
    fwrite($myfile, $txt);
    $txt = "Jane Doe\n";
    fwrite($myfile, $txt);
    fclose($myfile);
?>
```

Observeu que hem escrit al fitxer "newfile.txt" dues vegades. Cada vegada que escrivim al fitxer, enviem la cadena \$txt que primer contenia "John Doe" i la segona "Jane Doe". Després d'haver acabat d'escriure, hem tancat el fitxer mitjançant la fclose(). Comprova el codi.

Ara que "newfile.txt" conté algunes dades, podem mostrar què passa quan obrim un fitxer existent per escriure. Totes les dades existents s'esborraran i comencem amb un fitxer buit.

A l'exemple següent obrim el nostre fitxer existent "newfile.txt" i hi escrivim algunes dades noves:

```
<?php
    $myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
    $txt = "Mickey Mouse\n";
    fwrite($myfile, $txt);
    $txt = "Minnie Mouse\n";
    fwrite($myfile, $txt);
    fclose($myfile);
?>
```

Si ara obrim el fitxer "newfile.txt", tant en John com en Jane han desaparegut, i només hi ha les dades que acabem d'escriure. Comprova-ho.

Podeu afegir dades a un fitxer utilitzant el mode "a". El mode "a" afegeix text al final del fitxer, mentre que el mode "w" anul·la (i esborra) el contingut antic del fitxer.

A l'exemple següent obrim el nostre fitxer existent "newfile.txt" i hi afegim text:

```
<?php
    $myfile = fopen("newfile.txt", "a") or die("Unable to open file!");
    $txt = "Donald Duck\n";
    fwrite($myfile, $txt);
    $txt = "Goofy Goof\n";
    fwrite($myfile, $txt);
    fclose($myfile);
?>
```

Pujar fitxers al servidor

Amb PHP, és fàcil pujar fitxers al servidor. I amb la facilitat ve el perill, així que sempre aneu amb compte quan permeteu pujar fitxers!

Configureu el fitxer "php.ini". En primer lloc, assegureu-vos que PHP estiga configurat per a permetre la càrrega de fitxers.

Al fitxer "php.ini", cerqueu la directiva file_uploads i configureu-la a On: file_uploads = On

A continuació, creeu un formulari HTML que permeti als usuaris triar el fitxer d'imatge que volen carregar:

```
<!DOCTYPE html>
<html>
<body>

    <form action="upload.php" method="post" enctype="multipart/form-data">
        Select image to upload:
        <input type="file" name="fileToUpload" id="fileToUpload">
        <input type="submit" value="Upload Image" name="submit">
    </form>

</body>
</html>
```

Algunes regles a seguir per al formulari HTML anterior:

- Assegureu-vos que el formulari utilitza el method="post"
- El formulari també necessita l'atribut següent: enctype="multipart/form-data". Especifica quin tipus de contingut s'ha d'utilitzar en enviar el formulari

Sense els requisits anteriors, la càrrega del fitxer no funcionarà.

Altres coses a tenir en compte:

L'atribut type="file" de l'etiqueta <input> mostra el camp d'entrada com a control de selecció de fitxers, amb un botó "Navega" al costat del control d'entrada

El formulari anterior envia dades a un fitxer anomenat "upload.php", que crearem a continuació.

Creeu l'script PHP de càrrega del fitxer. El fitxer "upload.php" conté el codi per pujar un fitxer:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    }
}
```

```

    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
?>

```

Script PHP explicat:

\$target_dir = "càrregues/" - especifica el directori on es col·locarà el fitxer
 \$target_file especifica la ruta del fitxer que s'ha de carregar
 \$uploadOk=1 encara no s'utilitza (s'utilitzarà més tard)
 \$imageFileType conté l'extensió del fitxer (en minúscules)

Nota: Haureu de crear un nou directori anomenat "càrregues" al directori on resideix el fitxer "upload.php". Els fitxers penjats es desaran allà.

Comproveu si el fitxer ja existeix. Primer, comprovarem si el fitxer ja existeix a la carpeta "càrregues". Si ho fa, es mostra un missatge d'error i \$uploadOk s'estableix en 0:

```

// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}

```

Limitar la mida del fitxer. El camp d'entrada del fitxer al nostre formulari HTML anterior s'anomena "fitxerToUpload".

Ara volem comprovar la mida del fitxer. Si el fitxer és més gran que 500 KB, es mostra un missatge d'error i \$uploadOk s'estableix en 0:

```

// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}

```

Limitar el tipus de fitxer. El codi següent només permet als usuaris penjar fitxers JPG, JPEG, PNG i GIF. Tots els altres tipus de fitxer donen un missatge d'error abans de configurar \$uploadOk a 0:

```

// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType !=
"jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}

```

Script PHP de càrrega del fitxer complet

El fitxer complet "upload.php" ara té aquest aspecte:

```

<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;

```



```

$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}

// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}

// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}

// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType !=
"jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
    // if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". htmlspecialchars( basename( $_FILES["fileToUpload"]
["name"])) . " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>

```

4. Cookies

Què és una cookie?

Sovint s'utilitza una cookie per identificar un usuari. Una cookie és un xicotet fitxer que el servidor incrusta a l'ordinador de l'usuari. Cada vegada que el mateix ordinador sol·licite una pàgina amb un navegador, també enviarà la galeta. En PHP, podeu crear i recuperar valors de galetes.

Crea galetes

Es crea una galeta amb la funció `setcookie()`.

Sintaxi: ***setcookie(name, value, expire, path, domain, secure, httponly);***

Només cal el paràmetre de nom . Tots els altres paràmetres són opcionals.

Crea/recupera una cookie

L'exemple següent crea una galeta anomenada "usuari" amb el valor "John Doe". La galeta caducarà al cap de 30 dies (86400 * 30). El "/" significa que la galeta està disponible a tot el lloc web (en cas contrari, seleccioneu el directori que preferiu).

A continuació, recuperem el valor de la galeta "usuari" (utilitzant la variable global \$_COOKIE). També fem servir la funció isset() per a saber si la galeta està configurada:

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1
day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Nota: La funció setcookie() ha d'aparèixer ABANS de l'etiqueta <html>.

Nota: el valor de la galeta es codifica automàticament per URL quan s'envia la galeta i es descodifica automàticament quan es rep (per evitar la codificació d'URL, utilitzeu-la funció setrawcookie() en el seu lloc).

Modificar un valor de cookie

Per a modificar una galeta, només cal que torneu a configurar la galeta mitjançant la funció setcookie():

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
```

```
}  
?>
```

```
</body>  
</html>
```

Eliminar una cookie

Per a eliminar una galeta, utilitzeu la funció `setcookie()` amb una data de caducitat en el passat:

```
<?php  
// set the expiration date to one hour ago  
setcookie("user", "", time() - 3600);  
?>  
<html>  
<body>
```

```
<?php  
echo "Cookie 'user' is deleted.";  
?>
```

```
</body>  
</html>
```

Comproveu si les galetes estan habilitades

L'exemple següent crea un xicotet script que verifica si les galetes estan habilitades. Primer, proveu a crear una galeta de prova amb la funció `setcookie()` i, a continuació, comprova la variable `$_COOKIE`:

```
<?php  
setcookie("test_cookie", "test", time() + 3600, '/');  
?>  
<html>  
<body>
```

```
<?php  
if(count($_COOKIE) > 0) {  
    echo "Cookies are enabled.";  
} else {  
    echo "Cookies are disabled.";  
}  
?>
```

```
</body>  
</html>
```

5. Sessions

Una sessió és una manera d'emmagatzemar informació (en variables) per utilitzar-la en diverses pàgines. A diferència d'una cookie, la informació no s'emmagatzema a l'ordinador de l'usuari.

Què és una sessió PHP?

Quan treballem amb una aplicació, l'obriu, feu alguns canvis i després la tanqueu. Això és molt semblant a una sessió. L'ordinador sap qui eres. Sap quan iniciu l'aplicació i quan la

finalitzeu. Però a Internet hi ha un problema: el servidor web no sap qui ets ni què fas, perquè l'adreça HTTP no manté l'estat.

Les variables de sessió resolen aquest problema emmagatzemant la informació de l'usuari per utilitzar-la en diverses pàgines (per exemple, nom d'usuari, color preferit, etc.). Per defecte, les variables de sessió duren fins que l'usuari tanca el navegador.

Les variables de sessió contenen informació sobre un sol usuari i estan disponibles per a totes les pàgines d'una aplicació.

Consell: si necessiteu un emmagatzematge permanent, és possible que vulgueu emmagatzemar les dades en una base de dades .

Inicieu una sessió

S'inicia una sessió amb la funció `session_start()`. Les variables de sessió s'estableixen amb la variable global: `$_SESSION`.

Ara, anem a crear una pàgina nova anomenada "demo_session1.php". En aquesta pàgina, iniciem una nova sessió de PHP i establim algunes variables de sessió:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Nota: la funció `session_start()` ha de ser la primera cosa del vostre document. Abans de qualsevol etiqueta HTML.

Obteniu valors de variables de sessió PHP

A continuació, creem una altra pàgina anomenada "demo_session2.php". Des d'aquesta pàgina, accedirem a la informació de sessió que establim a la primera pàgina ("demo_session1.php").

Tingueu en compte que les variables de sessió no es passen individualment a cada pàgina nova, sinó que es recuperen de la sessió que obrim al començament de cada pàgina (`session_start()`).

Tingueu en compte també que tots els valors de les variables de sessió s'emmagatzemen a la variable global `$_SESSION`:

```
<?php
session_start();
```

```

?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>

```

Una altra manera de mostrar tots els valors de les variables de sessió per a una sessió d'usuari és executar el codi següent:

```

<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>

```

Com funciona? Com sap que sóc jo?

La majoria de sessions estableixen una clau d'usuari a l'ordinador de l'usuari que s'assembla a això: 765487cf34ert8dede5a562e4f3a7e12. Aleshores, quan s'obre una sessió en una altra pàgina, escaneja l'ordinador per trobar una clau d'usuari. Si hi ha una coincidència, accedeix a aquesta sessió, si no, inicia una nova sessió.

Modificar una variable de sessió PHP

Per a canviar una variable de sessió, només cal sobreescriure-la:

```

<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>

```

Destrueix una sessió PHP

Per eliminar totes les variables globals de sessió i destruir la sessió, utilitzeu `session_unset()` i `session_destroy()`:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

6. Filtres

Validació de dades = Determinar si les dades estan en la forma adequada.

Sana data = Elimina qualsevol caràcter il·legal de les dades.

L'extensió de filtres

Els filtres s'utilitzen per a validar i desinfectar l'entrada externa. L'extensió de filtres té moltes de les funcions necessàries per comprovar l'entrada de l'usuari i està dissenyada per a fer que la validació de dades siga més fàcil i ràpida.

La funció `filter_list()` es pot utilitzar per enumerar el que ofereix l'extensió de filtre PHP:

```
<!DOCTYPE html>
<html>
<head>
<style>
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
th, td {
    padding: 5px;
}
</style>
</head>
<body>

<table>
<tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
</tr>
<?php
foreach (filter_list() as $id =>$filter) {
    echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
}

```

```
?>
</table>

</body>
</html>
```

Per què utilitzar filtres?

Moltes aplicacions web reben entrada externa. Les entrades/dades externes poden ser:

- Entrada de l'usuari des d'un formulari
- Galetes
- Dades de serveis web
- Variables del servidor
- Resultats de la consulta a la base de dades

Sempre hauríeu de validar les dades externes! Les dades enviades no vàlides poden provocar problemes de seguretat i trencar la vostra pàgina web. Mitjançant l'ús de filtres PHP, podeu estar segur que la vostra aplicació rep l'entrada correcta!

Funció filter_var()

La funció filter_var() valida i desinfecta les dades. La funció filtra una variable amb un filtre especificat. Caldrà, doncs, dos paràmetres:

- La variable que voleu comprovar
- El tipus de xec a utilitzar

En l'exemple següent utilitza filter_var() per a eliminar totes les etiquetes HTML d'una cadena:

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Valida un nombre enter

L'exemple següent utilitza la funció filter_var() per a comprovar si la variable \$int és un nombre enter. Si \$int és un nombre enter, l'eixida serà: "El nombre sencer és vàlid". Si \$int no és un nombre enter, tindrem: "El nombre sencer no és vàlid":

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

A l'exemple anterior, si \$int es va establir en 0, la funció anterior retornarà "El nombre sencer no és vàlid". Per resoldre aquest problema, utilitzeu el codi següent:

```
<?php
```

```
$int = 0;

if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Valida una adreça IP

L'exemple següent utilitza la funció `filter_var()` per a comprovar si la variable `$ip` és una adreça IP vàlida:

```
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}
?>
```

Desinfecteu i valideu una adreça de correu electrònic

L'exemple següent utilitza la funció `filter_var()` per a eliminar primer tots els caràcters il·legals de la variable `$email` i després comprova si és una adreça de correu electrònic vàlida:

```
<?php
$email = "john.doe@example.com";

// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Desinfecteu i valideu una URL

L'exemple següent fa servir `filter_var()` per a eliminar primer tots els caràcters il·legals d'una URL i després comprova si `$url` és una URL vàlida:

```
<?php
$url = "https://www.w3schools.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
```



```
}  
?>
```

7. Funcions callback

Una funció de devolució de crida (funció callback) és una funció que es passa com a argument a una altra funció.

Qualsevol funció existent es pot utilitzar com a funció de devolució de crida. Per a utilitzar una funció com a funció de devolució de crida, passeu una cadena que continga el nom de la funció com a argument d'una altra funció:

Exemple: passeu una devolució de crida a la funció de PHP `array_map()` per a calcular la longitud de cada cadena d'una matriu:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
function my_callback($item) {  
    return strlen($item);  
}  
  
$strings = ["apple", "orange", "banana", "coconut"];  
$lengths = array_map("my_callback", $strings);  
print_r($lengths);  
?>  
  
</body>  
</html>
```

A partir de la versió 7, PHP pot passar funcions anònimes com a funcions de devolució de crida:

Exemple: utilitzeu una funció anònima com a devolució de crida per a la funció de PHP `array_map()`:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$strings = ["apple", "orange", "banana", "coconut"];  
$lengths = array_map( function($item) { return strlen($item); } , $strings);  
print_r($lengths);  
?>  
  
</body>  
</html>
```

Devolució de crida a les funcions definides per l'usuari

Les funcions i mètodes definits per l'usuari també poden prendre funcions de devolució de crida com a arguments. Per utilitzar funcions de devolució de crida dins d'una funció o mètode definit per l'usuari, crideu-lo afegint parèntesis a la variable i passeu arguments com amb les funcions normals:

Exemple: executeu una devolució de crida des d'una funció definida per l'usuari:

```

<!DOCTYPE html>
<html>
<body>

<?php
function exclaim($str) {
    return $str . "! ";
}

function ask($str) {
    return $str . "? ";
}

function printFormatted($str, $format) {
    // Calling the $format callback function
    echo $format($str);
}

// Pass "exclaim" and "ask" as callback functions to printFormatted()
printFormatted("Hello world", "exclaim");
printFormatted("Hello world", "ask");
?>

</body>
</html>

```

8. JSON

Què és JSON?

JSON significa JavaScript Object Notation i és una sintaxi per emmagatzemar i intercanviar dades.

Com que el format JSON és un format basat en text, es pot enviar fàcilment des d'un servidor i utilitzar-lo com a format de dades per qualsevol llenguatge de programació.

PHP i JSON

PHP té algunes funcions integrades per a gestionar JSON.

En primer lloc, veurem dues funcions:

- `json_encode()`
- `json_decode()`

`json_encode()`

La funció `json_encode()` s'utilitza per a codificar un valor en format JSON.

Aquest exemple mostra com codificar una matriu associativa i convertir-la en un objecte JSON:

```

<!DOCTYPE html>
<html>
<body>

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

```

```
var_dump(json_decode($jsonobj));
?>
```

```
</body>
</html>
```

Altre exemple: aquest exemple mostra com codificar una matriu indexada en una matriu JSON:

```
<!DOCTYPE html>
<html>
<body>

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj, true));
?>

</body>
</html>
```

json_decode()

La funció `json_decode()` s'utilitza per a descodificar un objecte JSON en un objecte PHP o una matriu associativa.

Aquest exemple descodifica les dades JSON en un objecte PHP:

```
<!DOCTYPE html>
<html>
<body>

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj));
?>

</body>
</html>
```

La funció `json_decode()` retorna un objecte per defecte. La funció `json_decode()` té un segon paràmetre `i`, quan s'estableix en `true`, els objectes JSON es descodifiquen en matrius associatives.

Aquest exemple descodifica les dades JSON en una matriu associativa PHP:

```
<!DOCTYPE html>
<html>
<body>

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj, true));
?>

</body>
</html>
```

Accés als valors descodificats

Ací hi ha dos exemples de com accedir als valors descodificats des d'un objecte i des d'una matriu associativa:

Aquest exemple mostra com accedir als valors des d'un objecte PHP:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

echo $obj->Peter;
echo $obj->Ben;
echo $obj->Joe;
?>
```

Aquest exemple mostra com accedir als valors des d'una matriu associativa:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

echo $arr["Peter"];
echo $arr["Ben"];
echo $arr["Joe"];
?>
```

Recorregut a través dels valors

També podeu recórrer els valors amb un bucle foreach() :

Aquest exemple mostra com passar pels valors d'un objecte PHP:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

foreach($obj as $key => $value) {
    echo $key . " => " . $value . "<br>";
}
?>
```

Aquest exemple mostra com fer un bucle a través dels valors d'una matriu associativa PHP:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

foreach($arr as $key => $value) {
    echo $key . " => " . $value . "<br>";
}
?>
```

9. Excepcions

Què és una excepció?

Una excepció és un objecte que descriu un error o un comportament inesperat d'un script PHP. Moltes classes i funcions PHP generen excepcions. Les funcions i classes definides per l'usuari també poden llançar excepcions.

Les excepcions són una bona manera de parar una funció quan es troba amb dades que no pot utilitzar.

Llançar una excepció

La instrucció `throw` permet a una funció o mètode definit per l'usuari llançar una excepció. Quan es produeix una excepció, el codi següent no s'executarà.

Si no es detecta una excepció, es produirà un error fatal amb un missatge "Excepció no detectada".

Intentem llançar una excepció sense captar-la:

```
<?php
function divide($dividend, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division by zero");
    }
    return $dividend / $divisor;
}

echo divide(5, 0);
?>
```

El resultat serà una cosa així:

```
Fatal error: Uncaught Exception: Division by zero in C:\webfolder\test.php:4
Stack trace: #0 C:\webfolder\test.php(9):
divide(5, 0) #1 {main} thrown in C:\webfolder\test.php on line 4
```

La declaració try...catch

Per evitar l'error de l'exemple anterior, podem utilitzar la instrucció `try...catch` per detectar excepcions i continuar el procés.

Sintaxi:

```
try {
    code that can throw exceptions
} catch(Exception $e) {
    code that runs when an exception is caught
}
```

Exemple: Mostra un missatge quan es produeix una excepció:

```
<?php
function divide($dividend, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division by zero");
    }
}
```

```

    return $dividend / $divisor;
}

try {
    echo divide(5, 0);
} catch(Exception $e) {
    echo "Unable to divide.";
}
?>

```

El bloc catch indica quin tipus d'excepció s'ha de detectar i el nom de la variable que es pot utilitzar per accedir a l'excepció. A l'exemple anterior, el tipus d'excepció és Exception i el nom de la variable és \$e.

La declaració try...catch...finally

La declaració try...catch...finally es pot utilitzar per detectar excepcions. El codi del bloc finally sempre s'executarà independentment de si s'ha detectat una excepció. Si finally està present, el bloc catch és opcional.

Sintaxi

```

try {
    code that can throw exceptions
} catch(Exception $e) {
    code that runs when an exception is caught
} finally {
    code that always runs regardless of whether an exception was caught
}

```

Mostra un missatge quan es produeix una excepció i després indica que el procés ha finalitzat:

```

<?php
function divide($dividend, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division by zero");
    }
    return $dividend / $divisor;
}

try {
    echo divide(5, 0);
} catch(Exception $e) {
    echo "Unable to divide. ";
} finally {
    echo "Process complete.";
}
?>

```

Emet una cadena fins i tot si no s'ha detectat una excepció:

```

<?php
function divide($dividend, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division by zero");
    }
    return $dividend / $divisor;
}

```

```
try {
    echo divide(5, 0);
} finally {
    echo "Process complete.";
}
?>
```

L'objecte d'excepció

L'objecte d'excepció conté informació sobre l'error o el comportament inesperat que ha trobat la funció.

Sintaxi

new Exception(message, code, previous)

Valors dels paràmetres

Parameter	Description
message	Optional. A string describing why the exception was thrown
code	Optional. An integer that can be used to easily distinguish this exception from others of the same type
previous	Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter

Mètodes

Quan es detecta una excepció, la taula següent mostra alguns dels mètodes que es poden utilitzar per obtenir informació sobre l'excepció:

Method	Description
getMessage()	Returns a string describing why the exception was thrown
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns null
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception

Exemple: informació d'eixida sobre una excepció:

```
<?php
function divide($dividend, $divisor) {
    if($divisor == 0) {
        throw new Exception("Division by zero", 1);
    }
    return $dividend / $divisor;
}

try {
    echo divide(5, 0);
} catch(Exception $ex) {
    $code = $ex->getCode();
    $message = $ex->getMessage();
    $file = $ex->getFile();
    $line = $ex->getLine();
    echo "Exception thrown in $file on line $line: [Code $code]"
}
```

```
$message";  
}  
?>
```

Tema 4: OOP



-
1. OOP
 2. Classes i objectes PHP
 3. Constructor
 4. Destructor
 5. Modificador d'accés
 6. Herència
 7. Constants
 8. Classes abstractes
 9. Interfícies
 10. Característiques
 11. Mètodes estàtics
 12. Propietats estàtiques
 13. Espais de noms
 14. Iterables

1. OOP

Què és OOP? Des de PHP5, també podeu escriure codi PHP en un estil orientat a objectes.

La programació orientada a objectes és més ràpida i fàcil d'executar.

OOP significa programació orientada a objectes.

La programació procedimental consisteix a escriure procediments o funcions que realitzen operacions sobre les dades, mentre que la programació orientada a objectes consisteix a crear objectes que continguin dades i funcions.

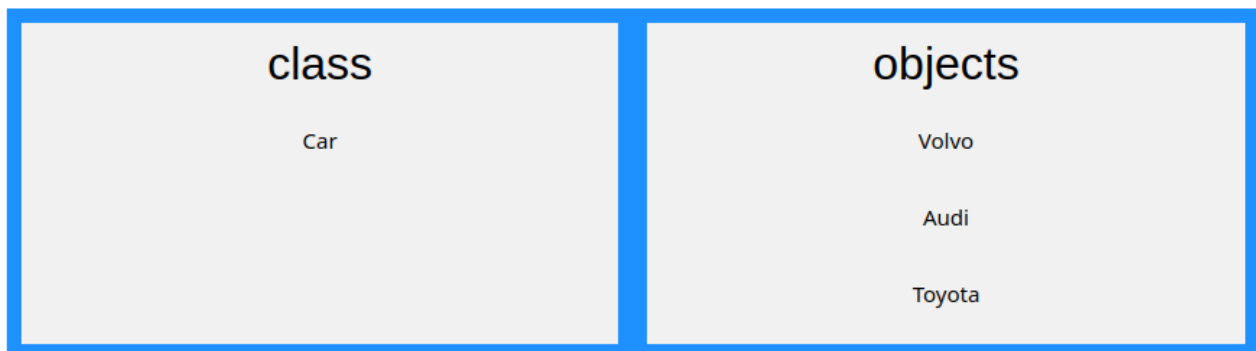
La programació orientada a objectes té diversos avantatges respecte a la programació procedimental:

- OOP és més ràpid i fàcil d'executar
- OOP proporciona una estructura clara per als programes
- OOP ajuda a mantenir el codi i fa que el codi siga més fàcil de mantenir, modificar i depurar
- La POO permet crear aplicacions reutilitzables completes amb menys codi i temps de desenvolupament més curt

Què són les classes i els objectes?

Les classes i els objectes són els dos aspectes principals de la programació orientada a objectes.

Mireu la il·lustració següent per veure la diferència entre classe i objectes:



Per tant, una classe és una plantilla per a objectes i un objecte és una instància d'una classe.

Quan es creen els objectes individuals, hereten totes les propietats i comportaments de la classe, però cada objecte tindrà valors diferents per a les propietats.

2. Classes i objectes PHP

Una classe és una plantilla per a objectes i un objecte és una instància de classe.

Suposem que tenim una classe anomenada Fruit. Una fruita pot tenir propietats com el nom, el color, el pes, etc. Podem definir variables com \$nom, \$color i \$pes per contenir els valors d'aquestes propietats.

Quan es creen els objectes individuals (poma, plàtan, etc.), hereten totes les propietats i comportaments de la classe, però cada objecte tindrà valors diferents per a les propietats.

Definir una classe

Una classe es defineix utilitzant la paraula clau class, seguida del nom de la classe i un parell de claus ({}). Totes les seues propietats i mètodes van dins de les claus:

Sintaxi

```
<?php
class Fruit {
    // code goes here...
}
?>
```

A continuació, declarem una classe anomenada Fruit que consta de dues propietats (\$name i \$color) i dos mètodes set_name() i get_name() per establir i obtenir la propietat \$name:

```
<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
}
?>
```

Nota: en una classe, les variables s'anomenen propietats i les funcions s'anomenen mètodes!

Definir objectes

Les classes no són res sense objectes! Podem crear diversos objectes a partir d'una classe. Cada objecte té totes les propietats i mètodes definits a la classe, però tindran valors de propietat diferents.

Els objectes d'una classe es creen mitjançant la paraula clau new.

A l'exemple següent, \$apple i \$banana són exemples de la classe Fruit:

```
<?php
class Fruit {
    // Properties
    public $name;
```

```

    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>

```

A l'exemple següent, afegim dos mètodes més a la classe Fruit, per configurar i obtenir la propietat \$color:

```

<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
    function set_color($color) {
        $this->color = $color;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>

```

La paraula clau \$this

La paraula clau \$this fa referència a l'objecte actual i només està disponible dins dels mètodes.

Mireu l'exemple següent:

```

<?php

```

```
class Fruit {  
    public $name;  
}  
$apple = new Fruit();  
?>
```

Aleshores, on podem canviar el valor de la propietat \$name? Hi ha dues maneres:

1. Dins de la classe (afegiu un mètode set_name() i utilitzeu \$this):

```
<?php  
class Fruit {  
    public $name;  
    function set_name($name) {  
        $this->name = $name;  
    }  
}  
$apple = new Fruit();  
$apple->set_name("Apple");  
  
echo $apple->name;  
?>
```

2. Fora de la classe (modificant directament el valor de la propietat):

```
<?php  
class Fruit {  
    public $name;  
}  
$apple = new Fruit();  
$apple->name = "Apple";  
  
echo $apple->name;  
?>
```

instanceof

Podeu utilitzar la paraula clau instanceof per a comprovar si un objecte pertany a una classe específica:

```
<?php  
$apple = new Fruit();  
var_dump($apple instanceof Fruit);  
?>
```

3. Constructor

Un constructor us permet inicialitzar les propietats d'un objecte en crear-lo.

Si creeu una funció __construct(), PHP cridarà automàticament aquesta funció quan creeu un objecte des d'una classe.

Observeu que la funció de construcció comença amb dos guions baixos (__):

Veiem a l'exemple següent, que utilitzar un constructor ens estalvia cridar al mètode set_name() que redueix la quantitat de codi:

Exemple

```

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>

```

Un altre exemple:

```

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    function get_name() {
        return $this->name;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>

```

4. Destructor

Es crida un destructor quan l'objecte es destrueix o l'script s'atura o surt.

Si creeu una funció `__destruct()`, PHP cridarà automàticament aquesta funció al final de l'script.

Observeu que la funció destrucció comença amb dos guions baixos (`__`)!

L'exemple següent té una funció `__construct()` que es crida automàticament quan creeu un objecte a partir d'una classe, i una funció `__destruct()` que es crida automàticament al final de l'script:

```

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name) {

```

```

        $this->name = $name;
    }
    function __destruct() {
        echo "The fruit is {$this->name}.";
    }
}

$apple = new Fruit("Apple");
?>

```

Un altre exemple:

```

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    function __destruct() {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

$apple = new Fruit("Apple", "red");
?>

```

5. Modificador d'accés

Les propietats i els mètodes poden tenir modificadors d'accés que controlen on s'hi pot accedir. Hi ha tres modificadors d'accés:

- **public**- Es pot accedir a la propietat o mètode des de qualsevol lloc. Això és per defecte
- **protected**- Es pot accedir a la propietat o mètode dins de la classe i per classes derivades d'aquesta classe
- **private**- NOMÉS es pot accedir a la propietat o mètode dins de la classe

A l'exemple següent hem afegit tres modificadors d'accés diferents a tres propietats (nom, color i pes). Així, si intenteu establir la propietat del nom, funcionarà bé (perquè la propietat del nom és pública i es pot accedir des de qualsevol lloc). Tanmateix, si intenteu establir la propietat de color o pes, es produirà un error fatal (perquè la propietat de color i pes està protegida i privada):

```

<?php
class Fruit {
    public $name;
    protected $color;
    private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>

```

En el següent exemple hem afegit modificadors d'accés a dues funcions. Així, si intenteu cridar a la funció `set_color()` o `set_weight()`, es produirà un error fatal (perquè les dues funcions es consideren protegides i privades), fins i tot si totes les propietats són públiques:

```
<?php
class Fruit {
    public $name;
    public $color;
    public $weight;

    function set_name($n) { // a public function (default)
        $this->name = $n;
    }
    protected function set_color($n) { // a protected function
        $this->color = $n;
    }
    private function set_weight($n) { // a private function
        $this->weight = $n;
    }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

6. Herència

Què és l'herència?

Herència en POO = Quan una classe deriva d'una altra classe.

La classe fill heretarà totes les propietats i mètodes públics i protegits de la classe pare. A més, pot tenir les seves pròpies propietats i mètodes.

Una classe heretada es defineix mitjançant la paraula clau `extends`.

Vegem un exemple:

```
<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    public function intro() {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
    public function message() {
        echo "Am I a fruit or a berry? ";
    }
}
```



```
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Exemple explicat

La classe Strawberry s'hereta de la classe Fruit.

Això vol dir que la classe Strawberry pot utilitzar les propietats públiques \$name i \$color, així com els mètodes públics __construct() i intro() de la classe Fruit a causa de l'herència.

La classe Strawberry també té el seu propi mètode: message().

Herència i modificador d'accés protegit

En el punt anterior vam aprendre que protected es pot accedir a propietats o mètodes dins de la classe i per classes derivades d'aquesta classe. Què vol dir això?

Vegem un exemple:

```
<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    protected function intro() {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

class Strawberry extends Fruit {
    public function message() {
        echo "Am I a fruit or a berry? ";
    }
}

// Try to call all three methods from outside class
$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is
public
$strawberry->message(); // OK. message() is public
$strawberry->intro(); // ERROR. intro() is protected
?>
```

A l'exemple anterior veiem que si intentem cridar un mètode protected (intro()) des de fora de la classe, rebrem un error. Amb public els mètodes funcionaran bé!

Vegem un altre exemple:

```
<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
}
```

```

        protected function intro() {
            echo "The fruit is {$this->name} and the color is {$this->color}.";
        }
    }

class Strawberry extends Fruit {
    public function message() {
        echo "Am I a fruit or a berry? ";
        // Call protected method from within derived class - OK
        $this-> intro();
    }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is
public
$strawberry->message(); // OK. message() is public and it calls intro() (which
is protected) from within the derived class
?>

```

A l'exemple anterior veiem que tot funciona bé! És perquè cridem el mètode protected (intro()) des de dins de la classe derivada.

Anul·lació de mètodes heretats

Els mètodes heretats es poden substituir redefinint els mètodes (utilitza el mateix nom) a la classe secundària.

Mireu l'exemple següent. Els mètodes __construct() i intro() de la classe secundària (Strawberry) anul·laran els mètodes __construct() i intro() de la classe pare (Fruit):

```

<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    public function intro() {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

class Strawberry extends Fruit {
    public $weight;
    public function __construct($name, $color, $weight) {
        $this->name = $name;
        $this->color = $color;
        $this->weight = $weight;
    }
    public function intro() {
        echo "The fruit is {$this->name}, the color is {$this->color}, and the
weight is {$this->weight} gram.";
    }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
?>

```

La paraula clau final

La paraula clau final es pot utilitzar per a evitar l'herència de classes o per evitar la substitució del mètode.

L'exemple següent mostra com prevenir l'herència de classes:

```
<?php
final class Fruit {
    // some code
}

// will result in error
class Strawberry extends Fruit {
    // some code
}
?>
```

L'exemple següent mostra com evitar la substitució del mètode:

```
<?php
class Fruit {
    final public function intro() {
        // some code
    }
}

class Strawberry extends Fruit {
    // will result in error
    public function intro() {
        // some code
    }
}
?>
```

7. Constants de classe

Les constants de classe poden ser útils si necessiteu definir algunes dades constants dins d'una classe.

Una constant de classe es declara dins d'una classe amb la paraula clau const. Una constant no es pot canviar un cop declarada. Les constants de classe distingeixen entre majúscules i minúscules. No obstant això, es recomana anomenar les constants amb totes les lletres majúscules.

Podem accedir a una constant des de fora de la classe utilitzant el nom de la classe seguit de l'operador de resolució d'àmbit (::) seguit del nom de la constant, com ací:

```
<?php
class Goodbye {
    const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

O bé, podem accedir a una constant des de dins de la classe utilitzant la paraula clau self seguida de l'operador de resolució d'àmbit (::) seguit del nom de la constant, com ací:

```
<?php
```

```

class Goodbye {
    const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
    public function byebye() {
        echo self::LEAVING_MESSAGE;
    }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>

```

8. Classes abstractes

Què són les classes i els mètodes abstractes?

Les classes i mètodes abstractes són quan la classe pare té un mètode amb nom, però necessita les seues classes fills per complir amb es tasques que ha de fer.

Una classe abstracta és una classe que conté almenys un mètode abstracte. Un mètode abstracte és un mètode que es declara, però no s'implementa al codi.

Una classe o mètode abstracte es defineix amb la paraula clau abstract:

Sintaxi

```

<?php
abstract class ParentClass {
    abstract public function someMethod1();
    abstract public function someMethod2($name, $color);
    abstract public function someMethod3(): string;
}
?>

```

Quan s'hereta d'una classe abstracta, el mètode de classe fill s'ha de definir amb el mateix nom i el mateix modificador d'accés o menys restringit. Per tant, si el mètode abstracte es defineix com a protegit, el mètode de classe fill s'ha de definir com a protegit o públic, però no privat. A més, el tipus i el nombre d'arguments necessaris han de ser els mateixos. Tanmateix, les classes filles poden tenir arguments opcionals.

Així, quan una classe secundària s'hereta d'una classe abstracta, tenim les regles següents:

- El mètode de classe fill s'ha de definir amb el mateix nom i torna a declarar el mètode abstracte pare
- El mètode de classe fill s'ha de definir amb el mateix modificador d'accés o amb un modificador d'accés menys restringit
- El nombre d'arguments necessaris ha de ser el mateix. Tanmateix, a més, la classe fill pot tenir arguments opcionals

Vegem un exemple:

```

<?php
// Parent class
abstract class Car {
    public $name;
    public function __construct($name) {

```

```

        $this->name = $name;
    }
    abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
    public function intro() : string {
        return "Choose German quality! I'm an $this->name!";
    }
}

class Volvo extends Car {
    public function intro() : string {
        return "Proud to be Swedish! I'm a $this->name!";
    }
}

class Citroen extends Car {
    public function intro() : string {
        return "French extravagance! I'm a $this->name!";
    }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>

```

Exemple explicat

Les classes Audi, Volvo i Citroen s'hereten de la classe Car. Això vol dir que les classes Audi, Volvo i Citroen poden utilitzar la propietat pública \$name així com el mètode públic __construct() de la classe Car a causa de l'herència.

Però, intro() és un mètode abstracte que s'hauria de definir a totes les classes fills i haurien de retornar una cadena.

Més exemples de classes abstractes

Vegem un altre exemple on el mètode abstracte té un argument:

```

<?php
abstract class ParentClass {
    // Abstract method with an argument
    abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
    public function prefixName($name) {
        if ($name == "John Doe") {
            $prefix = "Mr.";
        } elseif ($name == "Jane Doe") {

```

```

        $prefix = "Mrs.";
    } else {
        $prefix = "";
    }
    return "{$prefix} {$name}";
}
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>

```

Vegem un altre exemple on el mètode abstracte té un argument i la classe fill té dos arguments opcionals que no estan definits al mètode abstracte del pare:

```

<?php
abstract class ParentClass {
    // Abstract method with an argument
    abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
    // The child class may define optional arguments that are not in the parent's
    abstract method
    public function prefixName($name, $separator = ".", $greet = "Dear") {
        if ($name == "John Doe") {
            $prefix = "Mr";
        } elseif ($name == "Jane Doe") {
            $prefix = "Mrs";
        } else {
            $prefix = "";
        }
        return "{$greet} {$prefix}{$separator} {$name}";
    }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>

```

9. Interfícies

Què són les interfícies?

Les interfícies us permeten especificar quins mètodes ha d'implementar una classe.

Les interfícies faciliten l'ús d'una varietat de classes diferents de la mateixa manera. Quan una o més classes utilitzen la mateixa interfície, s'anomena "polimorfisme".

Les interfícies es declaren amb la paraula clau interface:

Sintaxi

```

<?php
interface InterfaceName {
    public function someMethod1();
    public function someMethod2($name, $color);
}

```

```

    public function someMethod3() : string;
}
?>

```

Interfícies vs. Classes abstractes

Les interfícies són similars a les classes abstractes. La diferència entre les interfícies i les classes abstractes són:

- Les interfícies no poden tenir propietats, mentre que les classes abstractes sí
- Tots els mètodes d'interfície han de ser públics, mentre que els mètodes de classe abstractes són públics o protegits
- Tots els mètodes d'una interfície són abstractes, de manera que no es poden implementar en codi i la paraula clau abstracta no és necessària
- Les classes poden implementar una interfície mentre hereten d'una altra classe alhora

Ús d'interfícies

Per implementar una interfície, una classe ha d'utilitzar la paraula clau implements.

Una classe que implementa una interfície ha d'implementar tots els mètodes de la interfície. Exemple:

```

<?php
interface Animal {
    public function makeSound();
}

class Cat implements Animal {
    public function makeSound() {
        echo "Meow";
    }
}

$animal = new Cat();
$animal->makeSound();
?>

```

A partir de l'exemple anterior, diguem que ens agradaria escriure un programari que gestione un grup d'animals. Hi ha accions que tots els animals poden fer, però cada animal ho fa a la seua manera.

Mitjançant interfícies, podem escriure algun codi que pot funcionar per a tots els animals, fins i tot si cada animal es comporta de manera diferent:

```

<?php
// Interface definition
interface Animal {
    public function makeSound();
}

// Class definitions
class Cat implements Animal {
    public function makeSound() {
        echo " Meow ";
    }
}

```

```

class Dog implements Animal {
    public function makeSound() {
        echo " Bark ";
    }
}

class Mouse implements Animal {
    public function makeSound() {
        echo " Squeak ";
    }
}

// Create a list of animals
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);

// Tell the animals to make a sound
foreach($animals as $animal) {
    $animal->makeSound();
}
?>

```

Exemple explicat

El gat, el gos i el ratolí són classes que implementen la interfície Animal, el que significa que tots són capaços de fer un so mitjançant el mètode makeSound(). Per això, podem recórrer tots els animals i dir-los que facen un so encara que no sabem quin tipus d'animal és cadascun.

Com que la interfície no indica a les classes com implementar el mètode, cada animal pot fer un so a la seua manera.

10. Característiques

Què són les característiques o trets?

PHP només admet l'herència única: una classe fill només pot heretar d'un sol pare.

Aleshores, què passa si una classe necessita heretar diversos comportaments? Els trets OOP resolen aquest problema.

Els trets s'utilitzen per declarar mètodes que es poden utilitzar en diverses classes. Els trets poden tenir mètodes i mètodes abstractes que es poden utilitzar en diverses classes, i els mètodes poden tenir qualsevol modificador d'accés (públic, privat o protegit).

Els trets es declaren amb la paraula clau trait:

Sintaxi

```

<?php
trait TraitName {
    // some code...
}
?>

```

Per utilitzar un tret en una classe, utilitzeu la useparaula clau:
Sintaxi


```
<?php
class MyClass {
    use TraitName;
}
?>
```

Vegem un exemple:

```
<?php
trait message1 {
    public function msg1() {
        echo "OOP is fun! ";
    }
}

class Welcome {
    use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

Exemple explicat

Ací, declarem un tret: missatge1. Després, creem una classe: Benvinguts. La classe utilitza el tret i tots els mètodes del tret estaran disponibles a la classe.

Si altres classes necessiten utilitzar la funció msg1(), simplement utilitzeu el tret message1 en aquestes classes. Això redueix la duplicació de codi, perquè no cal tornar a declarar el mateix mètode una i altra vegada.

Ús de múltiples trets

Vegem un altre exemple:

```
<?php
trait message1 {
    public function msg1() {
        echo "OOP is fun! ";
    }
}

trait message2 {
    public function msg2() {
        echo "OOP reduces code duplication!";
    }
}

class Welcome {
    use message1;
}

class Welcome2 {
    use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";
```

```
$obj2 = new Welcome2();  
$obj2->msg1();  
$obj2->msg2();  
?>
```

Exemple explicat

Ací, declarem dos trets: missatge1 i missatge2. A continuació, creem dues classes: Benvinguda i Benvinguda2. La primera classe (Welcome) utilitza el tret message1, i la segona classe (Welcome2) utilitza els trets missatge1 i missatge2 (múltiples trets estan separats per comes).

11. Mètodes estàtics

Els mètodes estàtics es poden cridar directament, sense crear primer una instància de la classe.

Els mètodes estàtics es declaren amb la paraula clau static:

```
<?php  
class ClassName {  
    public static function staticMethod() {  
        echo "Hello World!";  
    }  
}  
?>
```

Per accedir a un mètode estàtic, utilitzeu el nom de la classe, els dos punts dobles (::) i el nom del mètode:

Sintaxi

ClassName::staticMethod();

Vegem un exemple:

```
<?php  
class greeting {  
    public static function welcome() {  
        echo "Hello World!";  
    }  
}  
  
// Call static method  
greeting::welcome();  
?>
```

Exemple explicat

Ací, declarem un mètode estàtic: welcome(). A continuació, anomenem el mètode estàtic utilitzant el nom de la classe, dos punts dobles (::) i el nom del mètode (sense crear una instància de la classe primer).

Més informació sobre mètodes estàtics

Una classe pot tenir mètodes estàtics i no estàtics. Es pot accedir a un mètode estàtic des d'un mètode de la mateixa classe utilitzant la paraula clau self i els dos punts dobles (::):

```
<?php
class greeting {
    public static function welcome() {
        echo "Hello World!";
    }

    public function __construct() {
        self::welcome();
    }
}

new greeting();
?>
```

Els mètodes estàtics també es poden cridar des de mètodes d'altres classes. Per a fer-ho, el mètode estàtic hauria de ser públic:

```
<?php
class A {
    public static function welcome() {
        echo "Hello World!";
    }
}

class B {
    public function message() {
        A::welcome();
    }
}

$obj = new B();
echo $obj -> message();
?>
```

Per cridar un mètode estàtic des d'una classe secundària, utilitzeu la paraula clau `parent` dins de la classe secundària. Aquí, el mètode estàtic pot ser `public` o `protected`.

```
<?php
class domain {
    protected static function getWebsiteName() {
        return "W3Schools.com";
    }
}

class domainW3 extends domain {
    public $websiteName;
    public function __construct() {
        $this->websiteName = parent::getWebsiteName();
    }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

12. Propietats estàtiques

Les propietats estàtiques es poden cridar directament, sense crear una instància d'una classe.

Les propietats estàtiques es declaren amb la paraula clau `static`:

Sintaxi

```
<?php
class ClassName {
    public static $staticProp = "W3Schools";
}
?>
```

Per accedir a una propietat estàtica, utilitzeu el nom de la classe, els dos punts dobles (::) i el nom de la propietat:

Sintaxi

ClassName::\$staticProp;

Vegem un exemple:

```
<?php
class pi {
    public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

Exemple explicat

Ací, declarem una propietat estàtica: \$value. A continuació, fem ressò del valor de la propietat estàtica utilitzant el nom de la classe, dos punts dobles (::) i el nom de la propietat (sense crear una classe primer).

Més informació sobre propietats estàtiques

Una classe pot tenir propietats estàtiques i no estàtiques. Es pot accedir a una propietat estàtica des d'un mètode de la mateixa classe utilitzant la paraula clau self i els dos punts dobles (::):

```
<?php
class pi {
    public static $value=3.14159;
    public function staticValue() {
        return self::$value;
    }
}

$pi = new pi();
echo $pi->staticValue();
?>
```

Per cridar una propietat estàtica des d'una classe secundària, utilitzeu la paraula clau parent dins de la classe secundària:

```
<?php
class pi {
    public static $value=3.14159;
}

class x extends pi {
    public function xStatic() {
```

```

        return parent::$value;
    }
}

// Get value of static property directly via child class
echo x::$value;

// or get value of static property via xStatic() method
$x = new x();
echo $x->xStatic();
?>

```

13. Espais de noms

Els espais de noms són qualificadors que resolen dos problemes diferents:

- Permeten una millor organització agrupant classes que treballen juntes per realitzar una tasca
- Permeten utilitzar el mateix nom per a més d'una classe

Per exemple, podeu tenir un conjunt de classes que descriuen una taula HTML, com ara Taula, Fila i Cella, alhora que també teniu un altre conjunt de classes per descriure mobles, com ara Taula, Cadira i Llit. Els espais de noms es poden utilitzar per organitzar les classes en dos grups diferents alhora que evita que les dues classes Taula i Taula es barregen.

Declaració d'un espai de noms

Els espais de noms es declaren al principi d'un fitxer mitjançant la paraula clau namespace:

Declarar un espai de noms anomenat Html:

```

<?php
namespace Html;
?>

```

Nota: una declaració namespace ha de ser la primera cosa al fitxer PHP. El codi següent no seria vàlid:

```

<?php
echo "Hello World!";
namespace Html;
...
?>

```

Les constants, classes i funcions declarades en aquest fitxer pertanyeran a l' espai de noms HTML:

Exemple: creeu una classe de taula a l'espai de noms HTML:

```

<?php
namespace Html;
class Table {
    public $title = "";
    public $numRows = 0;
    public function message() {
        echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
    }
}

```

```

    }
}
$table = new Table();
$table->title = "My table";
$table->numRows = 5;
?>

<!DOCTYPE html>
<html>
<body>

<?php
$table->message();
?>

</body>
</html>

```

Per a més organització, és possible tenir espais de noms imbricats:

Declara un espai de noms anomenat Html dins d'un espai de noms anomenat Codi:

```

<?php
namespace Code\Html;
?>

```

Ús d'espais de noms

Qualsevol codi que segueix una declaració namespace està operant dins de l'espai de noms, de manera que les classes que pertanyen a l'espai de noms es poden crear una instància sense cap qualificador. Per accedir a classes des de fora d'un espai de noms, la classe ha de tenir l'espai de noms adjunt.

Exemple: utilitzeu classes de l'espai de noms HTML:

```

<?php
$table = new Html\Table();
$row = new Html\Row();
?>

```

Quan s'utilitzen moltes classes del mateix espai de noms al mateix temps, és més fàcil utilitzar la paraula clau namespace:

Exemple: utilitzeu classes de l'espai de noms HTML sense necessitat del qualificador Html:

```

<?php
namespace Html;
$table = new Table();
$row = new Row();
?>

```

Àlies d'espai de noms

Pot ser útil donar un àlies a un espai de noms o classe per facilitar-ne l'escriptura. Això es fa amb la paraula clau use:

Exemple: doneu un àlies a un espai de noms:

```

<?php

```

```
use Html as H;
$table = new H\Table();
?>
```

Exemple: doneu un àlies a una classe:

```
<?php
use Html\Table as T;
$table = new T();
?>
```

14. Iterables

Què és un iterable?

Un iterable és qualsevol valor que es pot recórrer amb un bucle `foreach()`.

El pseudotipus `iterable` es va introduir a PHP 7.1 i es pot utilitzar com a tipus de dades per a arguments de funció i valors de retorn de funció.

Ús de iterables

La paraula clau `iterable` es pot utilitzar com a tipus de dades d'un argument de funció o com a tipus de retorn d'una funció:

Exemple. Utilitzeu un argument de funció `iterable`:

```
<?php
function printIterable(iterable $myIterable) {
    foreach($myIterable as $item) {
        echo $item;
    }
}

$arr = ["a", "b", "c"];
printIterable($arr);
?>
```

Exemple. Retorna un `iterable`:

```
<?php
function getIterable():iterable {
    return ["a", "b", "c"];
}

$myIterable = getIterable();
foreach($myIterable as $item) {
    echo $item;
}
?>
```

Creació d'iterables

Totes les matrius són iterables, de manera que qualsevol matriu es pot utilitzar com a argument d'una funció que requereixi un `iterable`.

Un iterador conté una llista d'elements i proporciona mètodes per recórrer-los. Manté un punter a un dels elements de la llista. Cada element de la llista ha de tenir una clau que es pot utilitzar per trobar l'element.

Un iterador ha de tenir aquests mètodes:

- `current()` Retorna l'element al qual apunta actualment el punter. Pot ser qualsevol tipus de dades
- `key()` Retorna la clau associada a l'element actual de la llista. Només pot ser un nombre enter, flotant, booleà o cadena
- `next()` Mou el punter al següent element de la llista
- `rewind()` Mou el punter al primer element de la llista
- `valid()` Si el punter intern no apunta cap a cap element (per exemple, si es va cridar `next()` al final de la llista), això hauria de tornar false. Torna cert en qualsevol altre cas

Exemple. Implementeu la interfície Iterator i utilitzeu-la com a iterable:

```
<?php
// Create an Iterator
class MyIterator implements Iterator {
    private $items = [];
    private $pointer = 0;

    public function __construct($items) {
        // array_values() makes sure that the keys are numbers
        $this->items = array_values($items);
    }

    public function current() {
        return $this->items[$this->pointer];
    }

    public function key() {
        return $this->pointer;
    }

    public function next() {
        $this->pointer++;
    }

    public function rewind() {
        $this->pointer = 0;
    }

    public function valid() {
        // count() indicates how many items are in the list
        return $this->pointer < count($this->items);
    }
}

// A function that uses iterables
function printIterable(iterable $myIterable) {
    foreach($myIterable as $item) {
        echo $item;
    }
}

// Use the iterator as an iterable
$iterator = new MyIterator(["a", "b", "c"]);
```



```
printIterable($iterator);  
?>
```



-
1. Base de dades MySQL
 2. Connexió de PHP amb MySQL
 3. Crea una base de dades
 4. Crea una taula
 5. Inserció de dades
 6. Insereix múltiples
 7. Select Data
 8. where
 9. Ordena per
 10. Elimina dades
 11. Actualitza dades
 12. Límit de dades MySQL

1. Base de dades MySQL

En PHP, podeu connectar-vos i manipular bases de dades. MySQL és el sistema de bases de dades més popular utilitzat amb PHP.

Què és MySQL?

- MySQL és un sistema de bases de dades utilitzat a la web
- MySQL és un sistema de bases de dades que s'executa en un servidor
- MySQL és ideal tant per a aplicacions xicotetes com grans
- MySQL és molt ràpid, fiable i fàcil d'utilitzar
- MySQL utilitza SQL estàndard
- MySQL es compila en diverses plataformes
- MySQL es pot descarregar i utilitzar gratuïtament
- MySQL està desenvolupat, distribuït i suportat per Oracle Corporation
- MySQL porta el nom de la filla del cofundador Monty Widenius: My

Les dades d'una base de dades MySQL s'emmagatzemen en taules. Una taula és una col·lecció de dades relacionades i consta de columnes i files.

Les bases de dades són útils per emmagatzemar informació de manera categòrica. Una empresa pot tenir una base de dades amb les taules següents:

- Empleats
- Productes
- Clients
- Comandes

Sistema de base de dades PHP + MySQL

PHP combinat amb MySQL resulta una combinació multiplataforma.

Consultes de bases de dades

Una consulta és una pregunta o una petició.

Podem consultar una base de dades per obtenir informació específica i tornar un conjunt de registres.

Mireu la consulta següent (utilitzant SQL estàndard):

```
SELECT LastName FROM Employees
```

La consulta anterior selecciona totes les dades de la columna "Cognoms" de la taula "Empleats".

MySQL és el sistema de bases de dades estàndard de facto per a llocs web amb GRAN volums de dades i usuaris finals (com Facebook, Twitter i Viquipèdia).

2. Connexió de PHP amb MySQL

PHP 5 i posteriors poden funcionar amb una base de dades MySQL mitjançant:

- Extensió MySQLi (la "i" significa millorada)
- PDO (objectes de dades PHP)

Les versions anteriors de PHP utilitzaven l'extensió MySQL que va quedar obsoleta el 2012.

He d'utilitzar MySQLi o PDO?

Si necessiteu una resposta breu, seria "El que vulgueu".

Tant MySQLi com PDO tenen els seus avantatges:

PDO funcionarà en 12 sistemes de bases de dades diferents, mentre que MySQLi només funcionarà amb bases de dades MySQL.

Per tant, si heu de canviar el vostre projecte per utilitzar una altra base de dades, PDO facilita el procés. Només heu de canviar la cadena de connexió i unes quantes consultes. Amb MySQLi, haureu de reescriure el codi sencer, incloses les consultes.

Tots dos estan orientats a objectes, però MySQLi també ofereix una API de procediment.

Tots dos donen suport a les declaracions preparades. Les declaracions preparades protegeixen de la injecció SQL i són molt importants per a la seguretat de les aplicacions web.

Exemples de MySQL tant a MySQLi com a la sintaxi PDO

En aquest i en els següents punts mostrem tres maneres de treballar amb PHP i MySQL:

- MySQLi (orientat a objectes)
- MySQLi (procedimental)
- DOP

Obriu una connexió a MySQL

Abans de poder accedir a les dades de la base de dades MySQL, hem de poder connectar-nos al servidor:

Exemple (orientat a objectes de MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Exemple (procediment MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>

```

Exemple (DOP)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>

```

Nota: a l'exemple de PDO anterior també hem especificat una base de dades (myDB) . PDO requereix una base de dades vàlida per connectar-se. Si no s'especifica cap base de dades, es llança una excepció.

Consell: un gran avantatge de PDO és que té una classe d'excepció per gestionar qualsevol problema que es pugua produir a les consultes de la nostra base de dades. Si es llança una excepció dins del bloc try{ }, l'script deixa d'executar-se i flueix directament al primer bloc catch(){ }.

Tanqueu la connexió

La connexió es tancarà automàticament quan finalitza l'script. Per tancar la connexió abans, feu servir açò:

MySQLi orientat a objectes:

```
$conn->close();
```

MySQLi procedimental:

```
mysqli_close($conn);
```

DOP:

```
$conn = null;
```

3. Crea una base de dades

Una base de dades consta d'una o més taules.

Necessitareu privilegis especials CREATE per crear o suprimir una base de dades MySQL. Creeu una base de dades MySQL amb MySQLi i PDO

La instrucció CREATE DATABASE s'utilitza per crear una base de dades a MySQL.

Els exemples següents creen una base de dades anomenada "myDB":

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

Nota: Quan creeu una base de dades nova, només heu d'especificar els tres primers arguments de l'objecte mysqli (nom del servidor, nom d'usuari i contrasenya).

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}
```

```
mysqli_close($conn);  
?>
```

Exemple (DOP)

```
<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
  
try {  
    $conn = new PDO("mysql:host=$servername", $username, $password);  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
    $sql = "CREATE DATABASE myDBPDO";  
    // use exec() because no results are returned  
    $conn->exec($sql);  
    echo "Database created successfully<br>";  
} catch(PDOException $e) {  
    echo $sql . "<br>" . $e->getMessage();  
}  
  
$conn = null;  
?>
```

Consell: un gran avantatge de PDO és que té una classe d'excepció per gestionar qualsevol problema que es pugui produir a les consultes de la nostra base de dades. Si es llança una excepció dins del bloc try{ }, l'script deixa d'executar-se i flueix directament al primer bloc catch(){ }. Al bloc catch de dalt fem ressò de la instrucció SQL i del missatge d'error generat.

4. Crea una taula

Una taula de base de dades té el seu propi nom únic i consta de columnes i files.

La instrucció CREATE TABLE s'utilitza per crear una taula a MySQL.

Crearem una taula anomenada "MyGuests", amb cinc columnes: "id", "firstname", "lastname", "email" i "reg_date":

```
CREATE TABLE MyGuests (  
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
firstname VARCHAR(30) NOT NULL,  
lastname VARCHAR(30) NOT NULL,  
email VARCHAR(50),  
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP  
)
```

Notes a la taula anterior:

El tipus de dades especifica quin tipus de dades pot contenir la columna. Per obtenir una referència completa de tots els tipus de dades disponibles, aneu a la nostra referència de tipus de dades.

Després del tipus de dades, podeu especificar altres atributs opcionals per a cada columna:

- NOT NULL: cada fila ha de contenir un valor per a aquesta columna, no es permeten valors nuls
- Valor predeterminat: estableix un valor predeterminat que s'afegeix quan no es passa cap altre valor
- UNSIGNED: s'utilitza per a tipus de nombres, limita les dades emmagatzemades a nombres positius i zero
- INCREMENT AUTOMÀTIC: MySQL augmenta automàticament el valor del camp en 1 cada vegada que s'afegeix un registre nou
- CLAU PRIMÀRIA: s'utilitza per identificar de manera única les files d'una taula. La columna amb la configuració de CLAU PRIMÀRIA sol ser un número d'identificació i sovint s'utilitza amb AUTO_INCREMENT

Cada taula hauria de tenir una columna de clau primària (en aquest cas: la columna "id"). El seu valor ha de ser únic per a cada registre de la taula.

Els exemples següents mostren com crear la taula en PHP:

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
```



```

$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>

```

Exemple (DOP)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // sql to create table
    $sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Table MyGuests created successfully";
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>

```

5. Inserció de dades

Després de crear una base de dades i una taula, podem començar a afegir-hi dades.

Ací hi ha algunes regles de sintaxi a seguir:

- La consulta SQL s'ha de citar en PHP
- Els valors de cadena dins de la consulta SQL s'han de citar entre cometes
- Els valors numèrics no s'han de citar
- La paraula NULL no s'ha de citar

La instrucció INSERT INTO s'utilitza per afegir nous registres a una taula MySQL:

```
INSERT INTO nom_taula (columna1, columna2, columna3,...)
VALORS (valor1, valor2, valor3,...)
```

En el punt anterior vam crear una taula buida anomenada "MyGuests" amb cinc columnes: "id", "firstname", "lastname", "email" i "reg_date". Ara, omplim la taula amb dades.

Nota: si una columna és AUTO_INCREMENT (com la columna "id") o TIMESTAMP amb l'actualització predeterminada de current_timestamp (com la columna "reg_date"), no cal especificar-la a la consulta SQL. MySQL afegirà automàticament el valor.

Els exemples següents afegeixen un registre nou a la taula "Els meus convidats":

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
```

```

    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>

```

Exemple (DOP)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "New record created successfully";
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>

```

6. Insereix múltiples registres

Inseriu diversos registres a MySQL mitjançant MySQLi i PDO

S'han d'executar múltiples sentències SQL amb la funció `mysqli_multi_query()`.

Els exemples següents afegeixen tres registres nous a la taula "Els meus convidats":

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

```

```

}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com');";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>

```

Exemple (procediment MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com');";

if (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>

```

La forma de la DOP és una mica diferent:

Exemple (DOP)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}

```

```

// begin the transaction
$conn->beginTransaction();
// our SQL statements
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')");

// commit the transaction
$conn->commit();
echo "New records created successfully";
} catch(PDOException $e) {
// roll back the transaction if something failed
$conn->rollback();
echo "Error: " . $e->getMessage();
}

$conn = null;
?>

```

7. Select Data

Selecioneu Dades d'una base de dades MySQL. La instrucció SELECT s'utilitza per seleccionar dades d'una o més taules:

SELECT column_name(s) FROM table_name

o podem utilitzar el caràcter * per seleccionar TOTES les columnes d'una taula:

SELECT * FROM table_name

L'exemple següent selecciona les columnes id, nom i cognoms de la taula MyGuests i les mostra:

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
}

```

```

} else {
    echo "0 results";
}
$conn->close();
?>

```

Explicant l'exemple anterior:

Primer, configurem una consulta SQL que selecciona les columnes id, nom i cognoms de la taula MyGuests. La següent línia de codi executa la consulta i posa les dades resultants en a variable anomenada \$resultat.

Després, la funció num_rows() comprova si n'hi ha més de zero files retornades.

Si hi ha més de zero files retornades, el funció fetch_assoc() posa tots els resultats en una matriu associativa. Amb un bucle while imprimim els resultats.

L'exemple següent mostra el mateix que l'exemple anterior (procediment MySQLi):

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
?>

```

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

```

```

}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]." ".
$row["lastname"]."</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}
$conn->close();
?>

```

Seleccioneu dades amb PDO (+ declaracions preparades). L'exemple següent utilitza declaracions preparades.

Selecciona les columnes id, nom i cognoms de la taula MyGuests i ho mostra en una taula HTML:

Exemple (DOP)

```

<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
    function __construct($it) {
        parent::__construct($it, self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px solid black;'>" .
parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
    $stmt->execute();
}

```

```

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as
    $k=>$v) {
        echo $v;
    }
} catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>

```

8. where

Seleccioneu i filtreu dades d'una base de dades MySQL. La clàusula WHERE s'utilitza per filtrar registres.

La clàusula WHERE s'utilitza per extreure només aquells registres que compleixen la condició especificada.

SELECT column_name(s) FROM table_name WHERE column_name operator value

L'exemple següent selecciona les columnes id, nom i cognoms de la taula MyGuests on el cognom és "Doe" i el mostra a la pàgina:

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>

```

Explicant l'exemple anterior:

Primer, configurem la consulta SQL que selecciona les columnes id, nom i cognoms de la taula MyGuests on el cognom és "Doe". La següent línia de codi executa la consulta i posa les dades resultants en a variable anomenada \$resultat.

Aleshores la function num_rows() comprova si n'hi ha més de zero files retornades.

Si hi ha més de zero files retornades, la funció fetch_assoc() posa tots els resultats en una matriu associativa que podem fer un bucle while a través del conjunt de resultats i dóna eixida a les dades les columnes id, nom i cognoms.

L'exemple següent mostra el mateix que l'exemple anterior, de la manera procedimental de MySQLi:

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
?>
```

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
```

```

$result = $conn->query($sql);

if ($result->num_rows > 0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]." ".
$row["lastname"]."</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}
$conn->close();
?>

```

Selecioneu dades amb PDO (+ declaracions preparades)

Selecciona les columnes id, nom i cognoms de la taula MyGuests on el cognom és "Doe", i ho mostra en una taula HTML:

Exemple (DOP)

```

<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
    function __construct($it) {
        parent::__construct($it, self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px solid black;'>" .
parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests WHERE
lastname='Doe'");
    $stmt->execute();

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);

```

```

        foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as
        $k=>$v) {
            echo $v;
        }
    }
    catch(PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
    $conn = null;
    echo "</table>";
    ?>

```

9. Ordena per

Seleccioneu i ordeneu dades d'una base de dades MySQL

La clàusula ORDER BY s'utilitza per ordenar el conjunt de resultats de manera ascendent o descendent ordre.

La clàusula ORDER BY ordena els registres en ordre ascendent per defecte. Per ordenar els registres en ordre descendent, utilitzeu la paraula clau DESC.

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC/DESC

L'exemple següent selecciona les columnes id, nom i cognoms de MyGuests taula. Els registres estaran ordenats per la columna del cognom:

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>

```

Exemple (procedimental MySQLi)

```

<?php

```

```

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
        $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
?>

```

Exemple (orientat a objectes MySQLi)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]." ".
        $row["lastname"]."</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}
$conn->close();
?>

```

Seleccioneu dades amb PDO (+ declaracions preparades)

Ací seleccionem les columnes id, nom i cognoms de la taula MyGuests. Els registres estaran ordenats per la columna del cognom, i es mostrarà en una taula HTML:

Exemple (DOP)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
    function __construct($it) {
        parent::__construct($it, self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px solid black;'>" .
parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests ORDER BY
lastname");
    $stmt->execute();

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as
$k=>$v) {
        echo $v;
    }
} catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

10. Elimina dades

Suprimir dades d'una taula MySQL mitjançant MySQLi i PDO

La instrucció DELETE s'utilitza per eliminar registres d'una taula:

DELETE FROM table_name

WHERE some_column = some_value

Observeu la clàusula WHERE a la sintaxi DELETE: la clàusula WHERE especifica quin registre o registres s'han d'eliminar. Si ometeu la clàusula ON, tots els registres s'eliminaran!

Mirem la taula "MyGuests":

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

Els exemples següents supprimeixen el registre amb id=3 a la taula "Els meus convidats":

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";
```

```

if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>

```

Exemple (DOP)

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // sql to delete a record
    $sql = "DELETE FROM MyGuests WHERE id=3";

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Record deleted successfully";
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>

```

Després de suprimir el registre, la taula tindrà aquest aspecte:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

11. Actualitza dades

Actualitzar dades en una taula MySQL mitjançant MySQLi i PDO

La instrucció UPDATE s'utilitza per actualitzar els registres existents en una taula:

UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value

Observeu la clàusula WHERE a la sintaxi UPDATE: la clàusula WHERE especifica quin registre o registres s'han d'actualitzar. Si ometeu la clàusula ON, tots els registres s'actualitzaran!

Mirem la taula "MyGuests":

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15

2 Mary Moe mary@example.com 2014-10-23 10:22:30

Els exemples següents actualitzen el registre amb id=2 a la taula "MyGuests":

Exemple (orientat a objectes MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}

$conn->close();
?>
```

Exemple (procediment MySQLi)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql)) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Exemple (DOP)

```
<?php
$servername = "localhost";
$username = "username";
```



```

$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

    // Prepare statement
    $stmt = $conn->prepare($sql);

    // execute the query
    $stmt->execute();

    // echo a message to say the UPDATE succeeded
    echo $stmt->rowCount() . " records UPDATED successfully";
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>

```

Un cop actualitzat el registre, la taula tindrà aquest aspecte:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Doe	mary@example.com	2014-10-23 10:22:30

12. Límit de dades MySQL

Limiteu les seleccions de dades d'una base de dades MySQL

MySQL proporciona una clàusula LIMIT que s'utilitza per especificar el nombre de registres a tornar.

La clàusula LIMIT facilita la codificació de resultats de diverses pàgines o la paginació SQL, i és molt útil en taules grans. Tornar un gran nombre de registres pot tindre impacte en el rendiment.

Suposem que volem seleccionar tots els registres de l'1 al 30 (inclosos) d'una taula anomenats "Comandes". Aleshores, la consulta SQL es veuria així:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

Quan s'executa la consulta SQL anterior, retornarà els primers 30 registres.

Què passa si volem seleccionar els registres del 16 al 25 (inclosos)?

Mysql també proporciona una manera de gestionar-ho: utilitzant OFFSET.

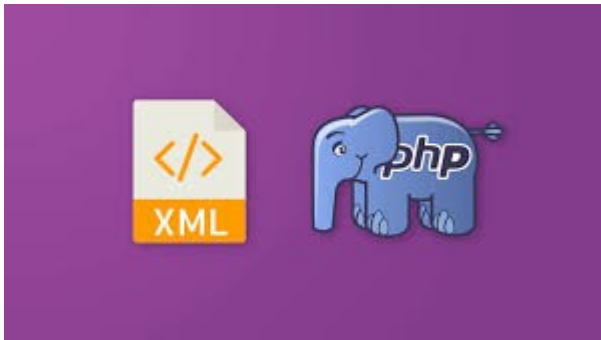
La consulta SQL següent diu "retorna només 10 registres, comença al registre 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

També podeu utilitzar una sintaxi més curta per aconseguir el mateix resultat:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Tema 6: PHP XML



-
1. Analitzadors PHP XML
 2. PHP SimpleXML Analitzador
 3. PHP SimpleXML - Get
 4. PHP XML Expat
 5. PHP XML DOM

1. Analitzadors PHP XML

Què és XML?

El llenguatge XML és una manera d'estructurar dades per compartir-les entre llocs web.

Diverses tecnologies web com els canals RSS i els podcasts estan escrites en XML.

XML és fàcil de crear. S'assembla molt a HTML, excepte que podeu crear les vostres pròpies etiquetes.

Què és un analitzador XML?

Per llegir i actualitzar, crear i manipular un document XML, necessitareu un analitzador XML.

A PHP hi ha dos tipus principals d'analitzadors XML:

- Analitzadors basats en arbres
- Analitzadors basats en esdeveniments

Analitzadors basats en arbres

Els analitzadors basats en arbres mantenen tot el document a la memòria i el transformen en un document XML amb estructura de l'arbre. Analitza tot el document i proporciona accés als elements de l'arbre (DOM).

Aquest tipus d'analitzador és una millor opció per als documents XML més xicotets, però no per a documents XML grans, ja que provoca importants problemes de rendiment.

Exemple d'analitzadors basats en arbres:

- SimpleXML
- DOM

Analitzadors basats en esdeveniments

Els analitzadors basats en esdeveniments no tenen tot el document a la memòria, sinó que llegeixen en un node i us permeten interactuar amb ells en temps real. Un cop passeu al següent node, l'antic ix de la memòria i es carrega el nou.

Aquest tipus d'analitzador és molt adequat per a documents XML grans. Analitza més ràpidament i consumeix menys memòria.

Exemple d'analitzadors basats en esdeveniments:

- XMLReader
- Analitzador d'expat XML

2. PHP SimpleXML Analitzador

SimpleXML és un analitzador basat en arbres. SimpleXML proporciona una manera senzilla d'obtenir el nom, els atributs i el text d'un element contingut si coneixeu l'estructura o la disposició del document XML.

SimpleXML converteix un document XML en una estructura de dades. En comparació amb DOM o l'analitzador d'expat, SimpleXML necessita menys línies de codi per a llegir dades de text d'un element.

Instal·lació

A partir de PHP 5, les funcions SimpleXML formen part del nucli de PHP. No es requereix cap instal·lació per utilitzar aquestes funcions.

PHP SimpleXML - Llegir des d'un string

La funció PHP `simplexml_load_string()` s'utilitza per llegir dades XML d'una cadena.

Suposem que tenim una variable que conté dades XML, com aquesta:

```
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";
```

L'exemple següent mostra com utilitzar la funció `simplexml_load_string()` per a llegir dades XML d'una cadena:

```
<?php
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";

$xml=simplexml_load_string($myXMLData) or die("Error: Cannot create object");
print_r($xml);
?>
```

L'eixida del codi anterior serà:

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder
[body] => Don't forget me this weekend! )
```

Consell de gestió d'errors: utilitzeu la funcionalitat `libxml` per recuperar tots els errors XML en carregar el document. L'exemple següent intenta carregar una cadena XML trencada:

```
<?php
libxml_use_internal_errors(true);
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
```

```

<document>
<user>John Doe</wronguser>
<email>john@example.com</wrongemail>
</document>";

$xml = simplexml_load_string($myXMLData);
if ($xml === false) {
    echo "Failed loading XML: ";
    foreach(libxml_get_errors() as $error) {
        echo "<br>", $error->message;
    }
} else {
    print_r($xml);
}
?>

```

L'eixida del codi anterior serà:

```

Failed loading XML:
Opening and ending tag mismatch: user line 3 and wronguser
Opening and ending tag mismatch: email line 4 and wrongemail

```

PHP SimpleXML - Llegir des de fitxer

La funció PHP `simplexml_load_file()` s'utilitza per a llegir dades XML d'un fitxer.

Suposem que tenim un fitxer XML anomenat " note.xml ", que es veu així:

```

<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

```

L'exemple següent mostra com utilitzar el `simplexml_load_file()` per a llegir dades XML d'un fitxer:

```

<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
print_r($xml);
?>

```

L'eixida del codi anterior serà:

```

SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder
[body] => Don't forget me this weekend! )

```

3. PHP SimpleXML - Get

SimpleXML és una extensió PHP que ens permet manipular i obtenir dades XML fàcilment.

PHP SimpleXML - Obteniu valors de nodes

Obteniu els valors dels nodes del fitxer " note.xml ":

```

<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");

```

```
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

La sortida del codi anterior serà:

```
Tove
Jani
Reminder
Don't forget me this weekend!
```

Suposem que tenim un fitxer XML anomenat " books.xml ", que es veu així:

```
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">XQuery Kick Start</title>
    <author>James McGovern</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

PHP SimpleXML - Obteniu valors de nodes d'elements específics

L'exemple següent obté el valor de l'element de node <title> al primer i els segons element <book> del fitxer "books.xml":

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
echo $xml->book[0]->title . "<br>";
echo $xml->book[1]->title;
?>
```

L'eixida del codi anterior serà:

```
Everyday Italian
Harry Potter
```

PHP SimpleXML - Obtenir valors de nodes - Loop

L'exemple següent mostra tots els elements <book> del fitxer "books.xml". i obté els valors dels elements de node <title>, <author>, <year> i <price>:

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title . ", ";
    echo $books->author . ", ";
    echo $books->year . ", ";
    echo $books->price . "<br>";
}
?>
```

L'eixida del codi anterior serà:

```
Everyday Italian, Giada De Laurentiis, 2005, 30.00
Harry Potter, J K. Rowling, 2005, 29.99
XQuery Kick Start, James McGovern, 2003, 49.99
Learning XML, Erik T. Ray, 2003, 39.95
```

PHP SimpleXML - Obteniu valors d'atributs

L'exemple següent obté el valor de l'atribut "categoria" del primer element <book> i el valor de l'atribut "lang" de l'element <title> al segon element <book>:

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
echo $xml->book[0]['category'] . "<br>";
echo $xml->book[1]->title['lang'];
?>
```

L'eixida del codi anterior serà:

```
COOKING
en
```

PHP SimpleXML - Obtenir valors d'atributs - Loop

L'exemple següent obté els valors d'atribut dels elements <title> al fitxer "books.xml":

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
foreach($xml->children() as $books) {
    echo $books->title['lang'];
    echo "<br>";
}
?>
```

L'eixida del codi anterior serà:

```
en
en
en-us
en-us
```

4. PHP XML Expat

El XML Expat Parser integrat permet processar documents XML en PHP.

L'analitzador XML Expat

L'analitzador Expat és un analitzador basat en esdeveniments.

Mireu aquesta línia XML:

```
<from>Jani</from>
```

Un analitzador basat en esdeveniments informa de l'XML anterior com una sèrie de tres esdeveniments:

- Element inicial: des de
- Inici secció CDATA, valor: Jani
- Element tancat: des de

Les funcions XML Expat Parser formen part del nucli de PHP. No cal instal·lació per a utilitzar aquestes funcions.

El fitxer XML

El fitxer XML "note.xml" s'utilitzarà a l'exemple següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Inicialització de l'analitzador XML Expat

Volem inicialitzar el XML Expat Parser en PHP, definir alguns controladors per a diferents Esdeveniments XML i, a continuació, analitzeu el fitxer XML.

```
<?php
// Initialize the XML parser
$parser=xml_parser_create();

// Function to use at the start of an element
function start($parser,$element_name,$element_attrs) {
    switch($element_name) {
        case "NOTE":
            echo "-- Note --<br>";
            break;
        case "TO":
            echo "To: ";
            break;
        case "FROM":
            echo "From: ";
            break;
        case "HEADING":
            echo "Heading: ";
            break;
        case "BODY":
            echo "Message: ";
    }
}
```

```

// Function to use at the end of an element
function stop($parser,$element_name) {
    echo "<br>";
}

// Function to use when finding character data
function char($parser,$data) {
    echo $data;
}

// Specify element handler
xml_set_element_handler($parser,"start","stop");

// Specify data handler
xml_set_character_data_handler($parser,"char");

// Open XML file
$fp=fopen("note.xml","r");

// Read data
while ($data=fread($fp,4096)) {
    xml_parse($parser,$data,feof($fp)) or
    die (sprintf("XML Error: %s at line %d",
    xml_error_string(xml_get_error_code($parser)),
    xml_get_current_line_number($parser)));
}

// Free the XML parser
xml_parser_free($parser);
?>

```

Exemple explicat:

- Inicialitzar l'analitzador XML amb la funció `xml_parser_create()`
- Creeu funcions per a utilitzar-les amb els diferents gestors d'esdeveniments
- Afegeix la funció `xml_set_element_handler()` per a especificar quina funció s'executarà quan l'analitzador es trobe amb les etiquetes d'obertura i tancament
- Afegeix la funció `xml_set_character_data_handler()` per a especificar quina funció s'executarà quan l'analitzador trobe dades de caràcters
- Analitzeu el fitxer "note.xml" amb la funció `xml_parse()`
- En cas d'error, afegir la funció `xml_error_string()` per convertir un error XML en una descripció textual
- Cridar a la funció `xml_parser_free()` per a alliberar la memòria assignada amb la funció `xml_parser_create()`

5. PHP XML DOM

L'analitzador DOM és un analitzador basat en arbres.

Mireu el següent fragment de document XML:

```

<?xml version="1.0" encoding="UTF-8"?>
<from>Jani</from>

```

El DOM veu l'XML anterior com una estructura d'arbre:

Nivell 1: Document XML

Nivell 2: element arrel: `<from>`

Nivell 3: element de text: "Jani"

Les funcions d'analitzador DOM formen part del nucli PHP. No **hi ha cap instal·lació necessària** per utilitzar aquestes funcions.

El fitxer XML

El fitxer XML següent ("note.xml") s'utilitzarà al nostre exemple:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Càrrega i eixida XML

Volem inicialitzar l'analitzador XML, carregar el xml i obtenir l'eixida:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

L'eixida del codi anterior serà:

```
Tove Jani Reminder Don't forget me this weekend!
```

Si seleccioneu "Mostra la font" a la finestra del navegador, veureu el següent HTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

L'exemple anterior crea un DOM i hi carrega l'XML de "note.xml".

Aleshores, la funció saveXML() col·loca el document XML intern en una cadena, de manera que el podem fer eixir.

Bucle a través de XML

Volem inicialitzar l'analitzador XML, carregar l'XML i recórrer tots els elements de l'element <note>:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
```

```
    print $item->nodeName . " = " . $item->nodeValue . "<br>";  
}  
?>
```

L'eixida del codi anterior serà:

```
#text =  
to = Tove  
#text =  
from = Jani  
#text =  
heading = Reminder  
#text =  
body = Don't forget me this weekend!  
#text =
```

A l'exemple anterior veieu que hi ha nodes de text buits entre cada element.

Quan es genera XML, sovint conté espais en blanc entre els nodes. L'analitzador XML DOM els tracta com a elements normals i de vegades causen problemes.



-
1. Introducció a AJAX
 2. AJAX PHP
 3. AJAX Base de dades
 4. AJAX XML
 5. AJAX Cerca en directe
 6. AJAX Enquesta

1. Introducció a AJAX

AJAX consisteix en actualitzar parts d'una pàgina web, sense tornar a carregar tota la pàgina.

Què és AJAX?

AJAX = JavaScript i XML asíncrons.

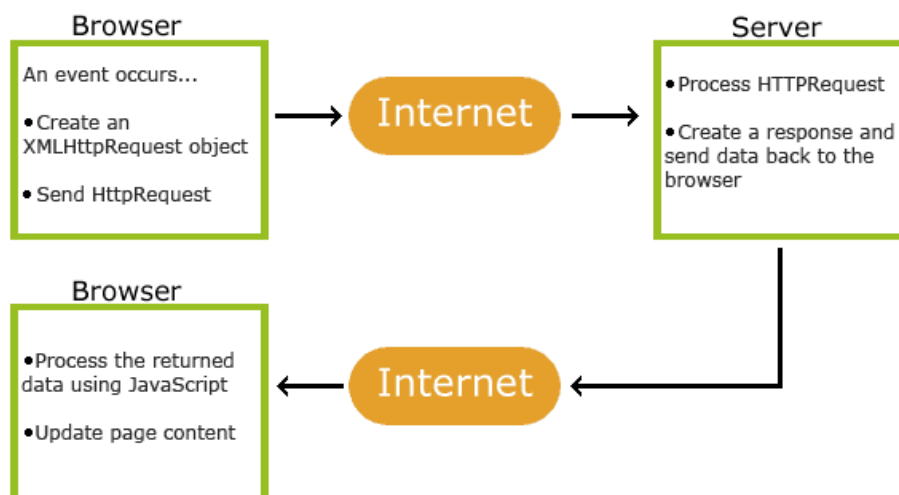
AJAX és una tècnica per crear pàgines web ràpides i dinàmiques.

AJAX permet que les pàgines web s'actualitzen de manera asíncrona intercanviant xicotetes quantitats de dades amb el servidor back end. Això vol dir que és possible actualitzar parts d'una pàgina web, sense tornar a carregar tota la pàgina.

Les pàgines web clàssiques, (que no utilitzen AJAX) han de tornar a carregar la pàgina sencera si el contingut ha de canviar.

Exemples d'aplicacions que utilitzen AJAX: Google Maps, Gmail, Youtube i Pestanyes de Facebook.

Com funciona AJAX



AJAX es basa en estàndards d'Internet i utilitza una combinació de:

- Objecte XMLHttpRequest (per intercanviar dades asíncronament amb un servidor)
- JavaScript/DOM (per mostrar/interactuar amb la informació)
- CSS (per estilitzar les dades)
- XML (sovint s'utilitza com a format per transferir dades)

Les aplicacions AJAX són independents del navegador i de la plataforma!

AJAX es va fer popular l'any 2005 per Google, amb Google Suggest.

Ara demostrarem com PHP i AJAX pot actualitzar parts d'una web pàgina, sense tornar a carregar tota la pàgina. L'script del servidor s'escriurà en PHP.

2. AJAX PHP

AJAX s'utilitza per crear aplicacions més interactives.

Exemple PHP AJAX

L'exemple següent mostrarà com una pàgina web es pot comunicar amb a servidor web mentre un usuari escriu caràcters en un camp d'entrada:

Exemple

Start typing a name in the input field below:

First name:

Suggestions: Petunia

Exemple explicat

A l'exemple anterior, quan un usuari escriu un caràcter al camp d'entrada, una funció anomenada *showHint()* s'executa.

La funció s'activa per l'esdeveniment *onkeyup*.

Ací teniu el codi HTML:

```
<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form action="">
    <label for="fname">First name:</label>
    <input type="text" id="fname" name="fname" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Explicació del codi:

Primer, comproveu si el camp d'entrada està buit (`str.length == 0`). Si és així, netegeu contingut del marcador de posició `txtHint` i eixiu de la funció.

Per contra, si el camp d'entrada no està buit, feu el següent:

- Crea un `XMLHttpRequest` objecte
- Creeu la funció que s'executarà quan la resposta del servidor estiga preparada
- Envieu la sol·licitud a un fitxer PHP (`gethint.php`) al servidor
- Para atenció que el paràmetre `q` s'afegeix a l'URL (`gethint.php?q=" + str`)
- I la variable `str` conté el contingut del camp d'entrada

El fitxer PHP - "gethint.php"

El fitxer PHP comprova una matriu de noms i retorna els noms corresponents. `gethint.php`:

```
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";

// lookup all hints from array if $q is different from ""
if ($q !== "") {
    $q = strtolower($q);
    $len=strlen($q);
    foreach($a as $name) {
```



```

    if (stristr($q, substr($name, 0, $len))) {
        if ($hint === "") {
            $hint = $name;
        } else {
            $hint .= ", $name";
        }
    }
}
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;

```

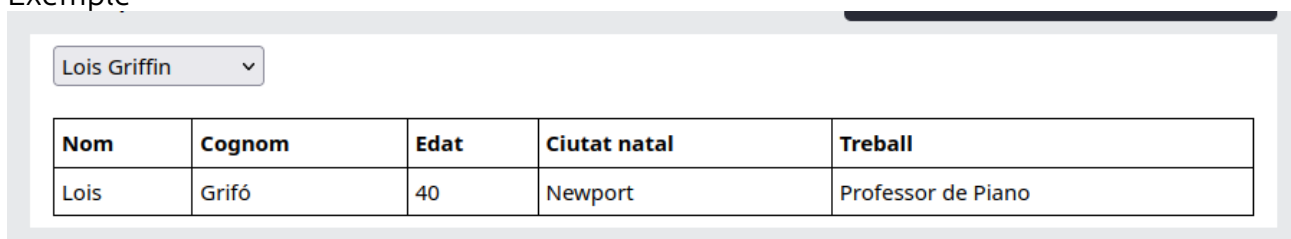
3. AJAX Base de dades

AJAX es pot utilitzar per a la comunicació interactiva amb una base de dades.

Exemple de base de dades AJAX

L'exemple següent mostrarà com una pàgina web pot obtenir informació d'una base de dades amb AJAX:

Exemple



The screenshot shows a web interface with a dropdown menu at the top containing the text 'Lois Griffin'. Below the menu is a table with the following data:

Nom	Cognom	Edat	Ciutat natal	Treball
Lois	Grifó	40	Newport	Professor de Piano

Exemple explicat: la base de dades MySQL

La taula de base de dades que fem servir a l'exemple anterior té aquest aspecte:

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

A l'exemple anterior, quan un usuari selecciona una persona a la llista desplegable a la part de dalt, una funció anomenada `showUser()` s'executa. La funció és activada per `onchange` esdeveniment.

Ací teniu el codi HTML:

```

<html>
<head>
<script>
function showUser(str) {
    if (str == "") {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "ajax_get.php?q=" + str, true);
        xmlhttp.send();
    }
}

```

```

    }
};
xmlhttp.open("GET", "family.php?q="+str,true);
xmlhttp.send();
}
}
</script>
</head>
<body>

<form>
<select name="users" onchange="showUser(this.value)">
  <option value="">Select a person:</option>
  <option value="1">Peter Griffin</option>
  <option value="2">Lois Griffin</option>
  <option value="3">Joseph Swanson</option>
  <option value="4">Glenn Quagmire</option>
</select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here...</b></div>

</body>
</html>

```

Explicació del codi:

Primer, comproveu si la persona està seleccionada. Si no s'ha seleccionat cap persona (str == ""), esborra l'opció contingut de la funció txtHinti d'eixida. Si es selecciona una persona, feu el següent:

- Crea un XMLHttpRequest objecte
- Creeu la funció que s'executarà quan la resposta del servidor estiga preparada
- Envieu la sol·licitud a un fitxer del servidor
- Observeu que un paràmetre q s'afix a l'URL (amb el contingut de la llista desplegable)

El fitxer PHP

La pàgina en el servidor a què crida el JavaScript anterior és un fitxer PHP anomenat family.php.

family.php executa una consulta a una base de dades MySQL i retorna el resultat en una taula HTML:

```

$q = intval($_GET['q']);

$con = mysqli_connect('localhost','peter','abc123');
if (!$con) {
    die('Could not connect: ' . mysqli_error($con));
}

mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);

echo "<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>

```

```

<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = mysqli_fetch_array($result)) {
    echo "<tr>";
    echo "<td>" . $row['FirstName'] . "</td>";
    echo "<td>" . $row['LastName'] . "</td>";
    echo "<td>" . $row['Age'] . "</td>";
    echo "<td>" . $row['Hometown'] . "</td>";
    echo "<td>" . $row['Job'] . "</td>";
    echo "</tr>";
}
echo "</table>";

mysqli_close($con);

```

Explicació: quan la consulta s'envia des del JavaScript al fitxer PHP, passa el següent:

- PHP obre una connexió a un servidor MySQL
- Es troba la persona correcta
- Es crea una taula HTML, s'omple de dades i es torna a enviar a txtHint en el client.

4. AJAX XML

AJAX es pot utilitzar per a la comunicació interactiva amb un fitxer XML.

Exemple XML AJAX

L'exemple següent mostrarà com una pàgina web pot obtenir informació d'un fitxer XML amb AJAX:

Exemple

Bob Dylan

TITLE: Empire Burlesque
ARTIST: Bob Dylan
COUNTRY: USA
COMPANY: Columbia
PRICE: 10.90
YEAR: 1985

Exemple explicat: la pàgina HTML

Quan un usuari selecciona un CD a la llista desplegable de dalt, s'executa una funció anomenada "showCD()". La funció s'activa per l'esdeveniment "onchange":

```

<html>
<head>
<script>
function showCD(str) {
    if (str=="") {
        document.getElementById("txtHint").innerHTML="";
    }
}

```

```

        return;
    }
    var xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
        if (this.readyState==4 && this.status==200) {
            document.getElementById("txtHint").innerHTML=this.responseText;
        }
    }
    xmlhttp.open("GET","getcd.php?q="+str,true);
    xmlhttp.send();
}
</script>
</head>
<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
    <option value="">Select a CD:</option>
    <option value="Bob Dylan">Bob Dylan</option>
    <option value="Bee Gees">Bee Gees</option>
    <option value="Cat Stevens">Cat Stevens</option>
</select>
</form>
<div id="txtHint"><b>CD info will be listed here...</b></div>

</body>
</html>

```

La funció showCD() fa el següent:

- Comproveu si s'ha seleccionat un CD
- Creeu un objecte XMLHttpRequest
- Creeu la funció que s'executarà quan la resposta del servidor estiga preparada
- Envieu la sol·licitud a un fitxer del servidor
- Observeu que s'afegeix un paràmetre (q) a l'URL (amb el contingut de la llista desplegable)

El fitxer PHP

La pàgina del servidor a què crida el JavaScript anterior és un fitxer PHP anomenat "getcd.php".

L'script PHP carrega un document XML, " cd_catalog.xml ", executa una consulta amb el fitxer XML i retorna el resultat com a HTML:

```

<?php
$q=$_GET["q"];

$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");

$x=$xmlDoc->getElementsByTagName('ARTIST');

for ($i=0; $i<=$x->length-1; $i++) {
    //Process only element nodes
    if ($x->item($i)->nodeType==1) {
        if ($x->item($i)->childNodes->item(0)->nodeValue == $q) {
            $y=($x->item($i)->parentNode);
        }
    }
}

```

```

    }
}

$cd=($y->childNodes);

for ($i=0;$i<$cd->length;$i++) {
    //Process only element nodes
    if ($cd->item($i)->nodeType==1) {
        echo("<b>" . $cd->item($i)->nodeName . ":</b> ");
        echo($cd->item($i)->childNodes->item(0)->nodeValue);
        echo("<br>");
    }
}
?>

```

Quan la consulta del CD s'envia des del JavaScript a la pàgina PHP, passarà acò:

- PHP crea un objecte XML DOM
- Cerca tots els elements <artist> que coincideixin amb el nom enviat des del JavaScript
- Mostra la informació de l'àlbum (envia al id "txtHint")

5. AJAX Cerca en directe (Live Search)

AJAX es pot utilitzar per crear cerques més fàcils d'utilitzar i interactives.

AJAX Live Search

L'exemple següent mostrarà una cerca en directe, on obteniu resultats de la cerca mentre escriviu.

La cerca en directe té molts avantatges en comparació amb la cerca tradicional:

- Els resultats es mostren a mesura que escriviu
- Els resultats es redueixen a mesura que continueu escrivint
- Si els resultats es tornen massa escassos, elimineu caràcters per veure un resultat més ampli

Si cerquem una pàgina de W3Schools en un camp d'entrada. Els resultats poden estar en un fitxer XML (links.xml).

Quan un usuari escriu un caràcter al camp d'entrada, s'executa la funció "showResult()". La funció s'activa amb "onkeyup" esdeveniment:

```

<html>
<head>
<script>
function showResult(str) {
    if (str.length==0) {
        document.getElementById("livesearch").innerHTML="";
        document.getElementById("livesearch").style.border="0px";
        return;
    }
    var xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
        if (this.readyState==4 && this.status==200) {
            document.getElementById("livesearch").innerHTML=this.responseText;
            document.getElementById("livesearch").style.border="1px solid #A5ACB2";

```

```

    }
}
xmlhttp.open("GET", "livesearch.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<form>
<input type="text" size="30" onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>

</body>
</html>

```

Explicació del codi font:

Si el camp d'entrada està buit (`str.length==0`), la funció esborra contingut del id de la cerca en directe i ix de la funció.

Si el camp d'entrada no està buit, la funció `showResult()` executa el següent:

- Creeu un objecte `XMLHttpRequest`
- Creeu la funció que s'executarà quan la resposta del servidor estiga preparada
- Envieu la sol·licitud a un fitxer del servidor
- Observeu que s'afegeix un paràmetre (`q`) a l'URL (amb el contingut del camp d'entrada)

El fitxer PHP

La pàgina del servidor a què crida el JavaScript anterior és un fitxer PHP anomenat "livesearch.php".

El codi font de "livesearch.php" cerca en un fitxer XML de títols que coincideixin amb la cadena de cerca i retorna el resultat:

```

<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

//get the q parameter from URL
$q=$_GET["q"];

//lookup all links from the xml file if length of q>0
if (strlen($q)>0) {
    $hint="";
    for($i=0; $i<($x->length); $i++) {
        $y=$x->item($i)->getElementsByTagName('title');
        $z=$x->item($i)->getElementsByTagName('url');
        if ($y->item(0)->nodeType==1) {
            //find a link matching the search text
            if (strstr($y->item(0)->childNodes->item(0)->nodeValue,$q)) {
                if ($hint=="") {
                    $hint="<a href='\" .
                    $z->item(0)->childNodes->item(0)->nodeValue .

```

```

        "' target='_blank'>" .
        $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    } else {
        $hint=$hint . "<br /><a href='" .
        $z->item(0)->childNodes->item(0)->nodeValue .
        "' target='_blank'>" .
        $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    }
}
}
}
}

// Set output to "no suggestion" if no hint was found
// or to the correct values
if ($hint=="") {
    $response="no suggestion";
} else {
    $response=$hint;
}

//output the response
echo $response;
?>

```

Si hi ha algun text enviat des de JavaScript (strlen(\$q) > 0), passa el següent:

- Carregueu un fitxer XML en un nou objecte XML DOM
- Recorre tots els elements <title> per trobar coincidències amb el text enviat des del JavaScript
- Estableix l'URL i el títol correctes a la variable "\$response". Si es troba més d'una coincidència, totes les coincidències s'afigen a la variable
- Si no es troben coincidències, la variable \$response s'estableix en "cap suggeriment"

6. AJAX Enquesta

L'exemple següent mostrarà una enquesta on es mostra el resultat sense tornar a carregar.

T'agrada PHP i AJAX fins ara?

Sí: ☐

No: ☐

Exemple explicat: la pàgina HTML

Quan un usuari tria una opció anterior, s'executa una funció anomenada "getVote()". La funció s'activa per l'esdeveniment "onclick":

```

<html>
<head>
<script>
function getVote(int) {
    var xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
        if (this.readyState==4 && this.status==200) {

```

```

        document.getElementById("poll").innerHTML=this.responseText;
    }
}
xmlhttp.open("GET", "poll_vote.php?vote="+int,true);
xmlhttp.send();
}
</script>
</head>
<body>

<div id="poll">
<h3>Do you like PHP and AJAX so far?</h3>
<form>
Yes: <input type="radio" name="vote" value="0"
onclick="getVote(this.value)"><br>
No: <input type="radio" name="vote" value="1" onclick="getVote(this.value)">
</form>
</div>

</body>
</html>

```

La funció getVote() fa el següent:

- Creeu un objecte XMLHttpRequest
- Creeu la funció que s'executarà quan la resposta del servidor estiga preparada
- Envieu la sol·licitud a un fitxer del servidor
- Observeu que s'afegeix un paràmetre (vote) a l'URL (amb el valor de l'opció sí o no)

El fitxer PHP

La pàgina del servidor a què crida el JavaScript anterior és un fitxer PHP anomenat "poll_vote.php":

```

<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0) {
    $yes = $yes + 1;
}
if ($vote == 1) {
    $no = $no + 1;
}

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>

```



```

<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>

```

El valor s'envia des de JavaScript i passarà açò:

- Obteniu el contingut del fitxer "poll_result.txt".
- Posa el contingut del fitxer en variables i afegiu-ne una a la variable seleccionada
- Escriu el resultat al fitxer "poll_result.txt".
- Emet una representació gràfica del resultat de l'enquesta

El fitxer de text

El fitxer de text (poll_result.txt) és on emmagatzemem les dades de l'enquesta.

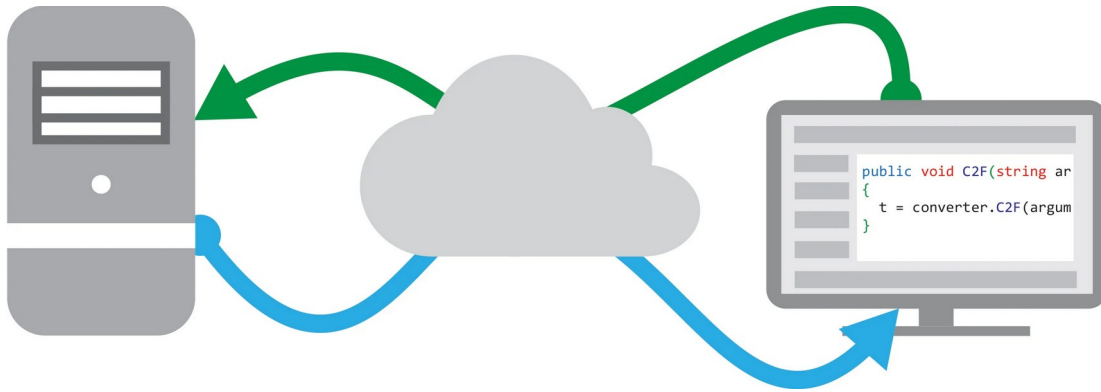
S'emmagatzema així:

0||0

El primer número representa els vots "Sí", el segon nombre representa els Vots "no".

Nota: recordeu que heu de permetre que el vostre servidor web edite el fitxer de text.

Tema 8: Crear un servei web i la seua interfície en PHP



-
1. Pas 1. Configuració de l'Entorn
 2. Pas 2. Crear el Servei Web RESTful
 3. Pas 3. Crear la Interfície d'Usuari
 4. Pas 4. Prova del Servei Web i la Interfície
 5. Servei SOAP

Crear un servei web i la seua interfície en PHP es pot dividir en diversos passos. Ací et guiaré a través d'un exemple bàsic per crear un servei web RESTful i la seva corresponent interfície.

Pas 1: Configuració de l'Entorn

Primer, assegura't de tenir un entorn PHP configurat. Pots utilitzar XAMPP, LAMP, o qualsevol altre servidor local per executar el teu codi PHP.

Pas 2: Crear el Servei Web RESTful

1- Estructura de Directoris:

```
/web_service
├── index.php
├── api
│   ├── db.php
│   ├── user.php
│   └── userController.php
```

2- Base de Dades (db.php):

Aquest fitxer conté la connexió a la base de dades.

```
<?php
class Database {
    private $host = "localhost";
    private $db_name = "testdb";
    private $username = "root";
    private $password = "";
    public $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" .
$this->db_name, $this->username, $this->password);
            $this->conn->exec("set names utf8");
        } catch (PDOException $exception) {
            echo "Error de connexió: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
?>
```

3- Model d'Usuari (user.php):

Aquest fitxer conté la classe d'usuari que interactua amb la base de dades.

```
<?php
class User {
    private $conn;
    private $table_name = "users";

    public $id;
    public $name;
    public $email;
```

```

        public $created;

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

        function read() {
            $query = "SELECT id, name, email, created FROM " . $this->table_name;
            $stmt = $this->conn->prepare($query);
            $stmt->execute();
            return $stmt;
        }
    }
?>

```

4- Controlador d'Usuari (UserController.php):

Aquest fitxer maneja les peticions al servei web.

```

<?php
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

include_once 'db.php';
include_once 'user.php';

$databse = new Database();
$db = $databse->getConnection();

$user = new User($db);

$stmt = $user->read();
$num = $stmt->rowCount();

if($num > 0) {
    $users_arr = array();
    $users_arr["records"] = array();

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        extract($row);
        $user_item = array(
            "id" => $id,
            "name" => $name,
            "email" => $email,
            "created" => $created
        );

        array_push($users_arr["records"], $user_item);
    }

    http_response_code(200);
    echo json_encode($users_arr);
} else {
    http_response_code(404);
    echo json_encode(array("message" => "No s'han trobat usuaris."));
}
?>

```

5- Fitxer Principal (index.php):

Aquest fitxer redirigeix les peticions als controladors adequats.

```
<?php
if (isset($_GET['action']) && $_GET['action'] == 'readUsers') {
    include 'api/userController.php';
} else {
    echo json_encode(array("message" => "Petició invàlida."));
}
?>
```

Pas 3: Crear la Interfície d'Usuari

Interfície HTML (index.html):

Aquest fitxer conté la interfície d'usuari que consumeix el servei web.

```
<!DOCTYPE html>
<html lang="ca">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Interfície del Servei Web</title>
    <script>
        function fetchUsers() {
            fetch('index.php?action=readUsers')
                .then(response => response.json())
                .then(data => {
                    let output = '<h2>Llista d\'Usuaris</h2>';
                    if (data.records) {
                        data.records.forEach(user => {
                            output += `
                                <ul>
                                    <li>ID: ${user.id}</li>
                                    <li>Nom: ${user.name}</li>
                                    <li>Correu electrònic:
${user.email}</li>
                                    <li>Creat: ${user.created}</li>
                                </ul>
                                `;
                        });
                    } else {
                        output = '<p>No s\'han trobat usuaris.</p>';
                    }
                    document.getElementById('output').innerHTML = output;
                })
                .catch(error => console.log('Error:', error));
        }
    </script>
</head>
<body>
    <button onclick="fetchUsers()">Obtenir Usuaris</button>
    <div id="output"></div>
</body>
</html>
```

Pas 4: Prova del Servei Web i la Interfície

Inicia el servidor local (per exemple, amb LAMP, Apache i MySQL).

Crea la base de dades i la taula d'usuaris:

```
CREATE DATABASE testdb;
USE testdb;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com'),
('Jane Doe', 'jane@example.com');
```

Accedeix a index.html en el teu navegador i fes clic a "Obtenir Usuaris" per veure les dades dels usuaris.

Amb aquests passos, has creat un servei web bàsic en PHP que ofereix dades d'usuaris i una interfície senzilla per consumir aquest servei.

5. Servei SOAP

SOAP (Simple Object Access Protocol) és un protocol de missatgeria per a l'intercanvi d'informació estructurada en la implementació de serveis web en xarxes informàtiques. SOAP utilitza el format XML per definir el missatge i s'envia a través de protocols com HTTP, SMTP, TCP, etc. Ací tens un resum de les seves característiques principals i com crear un servei web SOAP en PHP:

1- Característiques de SOAP

- Protocol Basat en XML: SOAP utilitza XML per formatar els missatges, assegurant la independència de la plataforma i del llenguatge.
- Protocol Agnòstic: SOAP pot ser utilitzat amb una varietat de protocols de transport, com ara HTTP, SMTP, TCP, etc.
- Estandarditzat: SOAP està estandarditzat per la W3C, la qual cosa garanteix la interoperabilitat entre sistemes diversos.
- Extensible: SOAP és extensible, el que permet afegir funcionalitats a través de capçaleres SOAP.

2- Crear un Servei Web SOAP en PHP

Per crear un servei web SOAP en PHP, seguirem els següents passos:

Configuració del Servidor PHP amb SOAP: Assegura't que l'extensió SOAP estiga activada en la teva instal·lació de PHP.

Crear el WSDL (Web Services Description Language): WSDL és un document XML que descriu els serveis web SOAP.

Implementar el Servei SOAP: Escriure el codi PHP que defineix les funcions del servei.

3- Crear un Client SOAP: Escriure el codi PHP que consumeix el servei SOAP.

Pas 1: Configuració del Servidor PHP

Assegura't que tens l'extensió SOAP activada en el teu fitxer php.ini:

```
extension=soap
```

Pas 2: Crear el Fitxer WSDL

service.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyService"
  targetNamespace="http://www.example.org/MyService/"
  xmlns:tns="http://www.example.org/MyService/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="getHelloRequest">
    <part name="name" type="xsd:string"/>
  </message>

  <message name="getHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="MyServicePortType">
    <operation name="getHello">
      <input message="tns:getHelloRequest"/>
      <output message="tns:getHelloResponse"/>
    </operation>
  </portType>

  <binding name="MyServiceBinding" type="tns:MyServicePortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getHello">
      <soap:operation
soapAction="http://www.example.org/MyService/getHello"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="MyService">
    <documentation>My Service</documentation>
    <port name="MyServicePort" binding="tns:MyServiceBinding">
      <soap:address location="http://localhost/web_service/service.php"/>
    </port>
  </service>
</definitions>
```

Pas 3: Implementar el Servei SOAP

service.php

```
<?php
class MyService {
    public function getHello($name) {
        return "Hola, " . $name;
    }
}
```

```
$options = array(
    'uri' => 'http://localhost/web_service/',
    'location' => 'http://localhost/web_service/service.php'
);

$server = new SoapServer('service.wsdl', $options);
$server->setClass('MyService');
$server->handle();
?>
```

Pas 4: Crear un Client SOAP

client.php

```
<?php
$options = array(
    'uri' => 'http://localhost/web_service/',
    'location' => 'http://localhost/web_service/service.php'
);

try {
    $client = new SoapClient('service.wsdl', $options);
    $result = $client->getHello('John');
    echo $result;
} catch (SoapFault $fault) {
    echo "Error: " . $fault->getMessage();
}
?>
```

4. Prova del Servei SOAP

- Inicia el servidor local (per exemple, amb LAMP, Apache).
- Accedeix a client.php en el teu navegador per veure la resposta del servei SOAP.

Amb aquests passos, has creat un servei web SOAP en PHP que ofereix una salutació i un client que consumeix aquest servei.

Tema 9: Laravel



-
1. Instal·lar Composer i Laravel a Ubuntu
 2. Introducció a Laravel

1. Instal·lar Composer i Laravel a Ubuntu

Per instal·lar Composer i Laravel a Ubuntu, segueix aquests passos detallats:
Instal·lació de Composer

Actualitzar el sistema:

```
sudo apt update  
sudo apt upgrade
```

Instal·lar les dependències necessàries:

```
sudo apt install curl php-cli php-mbstring git unzip
```

Descarregar i instal·lar Composer:

```
curl -sS https://getcomposer.org/installer -o composer-setup.php
```

Instal·lar Composer:

```
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

Verificar la instal·lació de Composer: `composer --version`

Instal·lació de Laravel. Instal·lar l'instal·lador de Laravel usant Composer:

```
composer global require laravel/installer
```

Afegir el directori de Composer al teu PATH:

Afegeix aquesta línia al final del teu fitxer `~/.bashrc` o `~/.zshrc`:

```
export PATH="$HOME/.config/composer/vendor/bin:$PATH"
```

Després, recarrega el fitxer:

```
source ~/.bashrc o source ~/.zshrc
```

Verificar la instal·lació de Laravel: `laravel --version`

Crear un nou projecte Laravel:

```
laravel new nom-del-projecte
```

Alternativament, pots usar Composer directament per crear un projecte Laravel:

```
composer create-project --prefer-dist laravel/laravel nom-del-projecte
```

Navegar al directori del projecte i servir l'aplicació:

```
cd nom-del-projecte  
php artisan serve --host=0.0.0.0 --port=8000
```

Ara, la teva aplicació Laravel estarà disponible a `http://localhost:8000`.

Seguint aquests passos, hauries de tenir Composer i Laravel instal·lats i funcionant al teu sistema Ubuntu.

2. Introducció a Laravel

Laravel és un dels frameworks PHP més populars per al desenvolupament d'aplicacions web. Va ser creat per Taylor Otwell amb l'objectiu de fer el desenvolupament web més ràpid i fàcil, proporcionant una arquitectura robusta i una varietat d'eines integrades. Laravel segueix el patró arquitectònic Model-View-Controller (MVC), que ajuda a separar la lògica de l'aplicació de la seva interfície d'usuari.

Característiques clau de Laravel

- Routing elegant: Laravel proporciona un sistema de rutes senzill i flexible que permet definir fàcilment les rutes de l'aplicació.
- Eloquent ORM: Un Object-Relational Mapper intuïtiu que facilita el treball amb bases de dades.
- Blade Templating Engine: Un motor de plantilles lleuger però potent que permet crear vistes dinàmiques.
- Middleware: Facilita la manipulació de les sol·licituds HTTP abans que arriben al controlador.
- Autenticació i autorització: Mecanismes integrats per gestionar l'autenticació d'usuaris i la seua autorització.
- Migracions: Faciliten la gestió de l'estructura de la base de dades.