**CLASE 01 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**PHP:** Es un lenguaje de código abierto muy popular, adecuado para desarrollo web y que puede ser incrustado en HTML.

**Popular** -> Muchas páginas están creadas con PHP / **Código** **abierto**: Uso libre y gratuito / **Incrustado**: Combinación con HTML.

Multiplataforma / Invisible al navegador / Capaz de conexión a base de datos / Capaz de expandir su potencial con plugins.

Servidores -> Ordenadores potentes con un software y hardware especial, que resuelven peticiones de otros ordenadores.

Clientes -> Cualquier dispositivo (PC, Notebook, Tablet) que generen peticiones hacia los servidores.

**¿Cómo funciona?** Cuando un cliente hace una petición, el servidor ejecuta el interprete de PHP (Para nuestro caso). Este compila el código fuente que genera el sitio web. El resultado es enviado al navegador del cliente.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ESTRUCTURA BASICA:**

**<?php** El codigo va en el medio. Con echo se muestra mensaje en el navegador. Todas las

echo “HOLA MUNDO”; instrucciones terminan en (;). La extension del archivo Fuente debe ser .php

**?>**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Variables: PHP soporta 8 tipos primitivos**

**Escalares**: Boolean – Integer – Float – String // **Compuestos**: Array – Object // **Especiales**: Resource – NULL.

Las variables comienzan con símbolo **$**. Son case sensitivity. Los tipos se definen cuando se les asigna valor.

Ejemplos: **$**nombre = “JUAN”; **$**edad = 25; **$**sueldo = 8500 // **echo** “edad:”, $edad;

Las conversiones las realiza PHP dependiendo del contenido de las variables. También se puede realizar una conversión explicita.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Funciones de cadenas:**

**strlen():** Retorna la cantidad de caracteres de una cadena.

**strcmp():** Compara dos cadenas (Teniendo en cuenta el case sensitive).

**strtolower():** Convierte la cadena a minúsculas.

**strtoupper():** Convierte la cadena a mayúsculas.

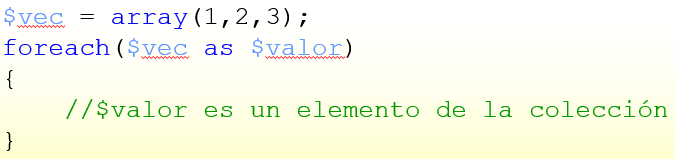
**substr():** Retorna una porción de la cadena.

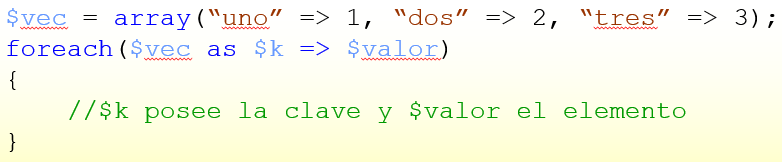
**ucfirst():** Convierte el primer carácter de la cadena a mayúsculas.

**ucwords():** Convierte el primer carácter de cada palabra de la cadena en mayúsculas.

**Operadores**: Aritméticos / Asignación / Comparación / Decremento / Lógicos / De Cadena / De Array.

**Sentencia Foreach:**





**ARRAYS:** Los Arrays en PHP son realmente mapas ordenados. Un mapa es un tipo de dato que asocia valores con claves.

**Arrays indexados** -> Índices numéricos **Arrays asociativos** -> Índices nombrados

**Arrays multidimensionales** -> Arrays que contienen otros Arrays dentro.

**Los Arrays pueden ser creados con el constructor del lenguaje array().**

$vec = array (1,2,3);

var\_dump($vec); **// array(3){[0]=>int(1) [1]=>int(2) [2]=>int(3)}**

**O sino simplemente:**

$vec[0] = 1;

$vec[1] = 2;

$vec[2] = 3;

var\_dump($vec); **// array(3){[0]=>int(1) [1]=>int(2) [2]=>int(3)}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Arrays Asociativos** con el constructor **array().**

$vec = array("Juan"=>22, "Romina"=>12, "Uriel"=>8);

var\_dump($vec); **// array(3){["Juan"]=>int(22)["Romina"]=>int(12)[" Uriel"]=>int(8)}**

$vec["Hugo"] =15;

$vec["Juana"] = 36;

var\_dump($vec); **// array(2){["Hugo"]=>int(15)["Juana"]=>int(36)}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Funciones para ordenar Arrays:**

**sort():** Ordena un array de forma ascendente.

**rsort():** Ordena un array de forma descendente.

**asort():** Ordena un array asociativo de forma ascendente por su valor.

**ksort():** Ordena un array asociativo de forma ascendente por su clave.

**arsort():** Ordena un array asociativo de forma descendente por su valor.

**krsort():** Ordena un array asociativo de forma descendente por su clave.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CLASE 02 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Programación del lado del servidor – Funciones propias:**

La declaración de la función comienza con la palabra **function**. El nombre de la función puede empezar con letra o guion bajo, pero no con números. Los nombres de las funciones NO son case sensitive.

**function NombreFuncion($par\_1, $par\_2,..., $par\_n){ //código }**

**Puede retornar valores. Puede tener parametros con valores por default.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INCLUIR ARCHIVOS:**

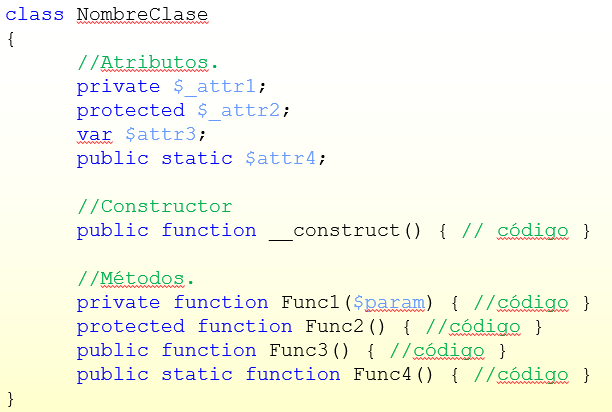
**Include o require:** Copia todo el código existente del archivo especificado dentro del archivo que posee dicha declaración.

**require**: Produce un error fatal (E\_COMPILE\_ERROR) y frenará el script

**include**: Produce una advertencia (E\_WARNING) y el script va a continuar.

**CLASES Y OBJETOS:**

**¿Cómo se declara una clase en PHP?**



**¿Cómo se declara un objeto?**

$**nombreObj** = new **NombreClase**();

Se utiliza el operador “**->”** para acceder a los miembros de instancia de la clase:

$nombreObj->Func3();

$nombreObj->attr3;

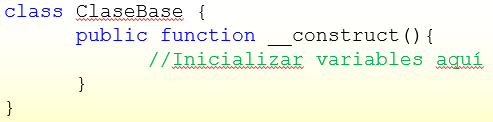
El operador “**::”** es utilizado para acceder a los miembros estaticos de la clase:

NombreClase::Func4();

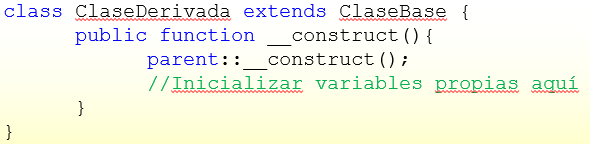
NombnreClase::$attr4;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

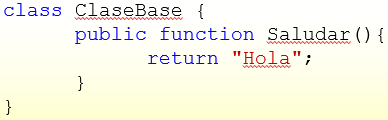
**HERENCIA: Se indica herencia a partir de extends.**

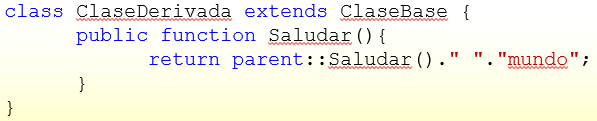


**CLASE DERIVADA:**

****

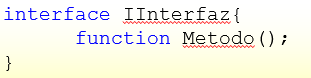
**POLIMORFISMO: Cualquier método puede ser modificado en sus clases derivadas:**

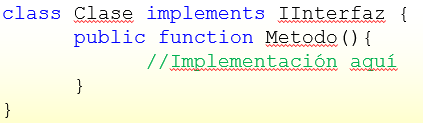




**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

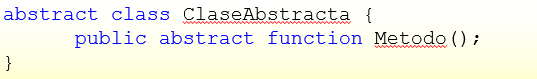
**INTERFACES: Las interfaces en PHP solo pueden contener declaraciones de métodos:**

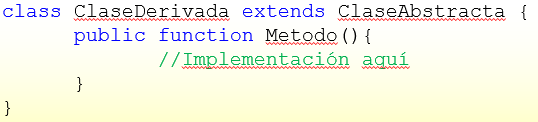




**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CLASES ABSTRACTAS Pueden contener atributos y métodos. Pero solo ellas pueden contener métodos con el modificador abstract.**





**CLASE 03 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Manipulación de archivos:**

**Abrir archivos:**  **int** fopen(archivo, **modo**);

**MODOS:**

r Abre un archivo para solo lectura. El cursor comienza al principio del archivo.

w Abre un archivo para solo escritura. Si no existe, crea uno nuevo. Si existe, borra el contenido.

a Abre un archivo para solo escritura. Si no existe, crea uno nuevo. Si existe, mantiene el contenido. Cursor al final.

x Crea un archivo para solo lectura. Retorna false y un error si el archivo existe.

r+ Abre un archivo para lectura / escritura. Idem r.

w+ Abre un archivo para lectura / escritura. Idem w.

a+ Abre un archivo para lectura / escritura. Idem a.

x+ Abre un archivo para lectura / escritura. Idem x.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CERRAR ARCHIVOS:**

**$ar** = fopen(archivo, modo);

fclose(**$ar**);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**LEER ARCHIVOS:**

**echo** fread(indicador\_archivo, filesize(archivo)); // Retorna un string con todo el contenido del archivo.

**string** fgets(indicador\_archivo); // Nos permite leer una linea de un archivo abierto.

**¿Cómo uso fgets para leer todo un archivo?**

**$ar** = fopen(archivo, modo);

**while**(!feof($ar)) {

**echo** fgets($ar); }

**fclose**($ar);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ESCRIBIR ARCHIVOS:**

**fwrite**: Nos permite escribir un archivo abierto. **fputs**: Nos permite escribir un archivo abierto.

**fwrite**(indicador\_archivo, texto [,longitud]); **fputs**(indicador\_archivo, texto [,longitud]);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**COPIAR ARCHIVOS:**

**copy**(archivo\_origen, archivo\_destino);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BORRAR ARCHIVOS:**

**unlink**(archivo);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ENVIO DE DATOS:**

**HTTP – Hypertext Transfer Protocol.**

Esta diseñado para permitir comunicación entre clientes y servidores. Funciona como un protocolo de pedido-respuesta. Un navegador web puede ser un cliente y una aplicación sobre un computador que aloja un sitio web puede ser el servidor.

**GET.**

El par de nombres / valores es enviado en la dirección URL (Texto claro). Las peticiones GET se pueden almacenar en caché. Permanecen en el historial del navegador. Pueden ser marcadas (book marked). Nunca debe ser utilizado cuando se trata de datos confidenciales. Tienen una longitud máxima de 2048 caracteres en la URL.

**POST.**

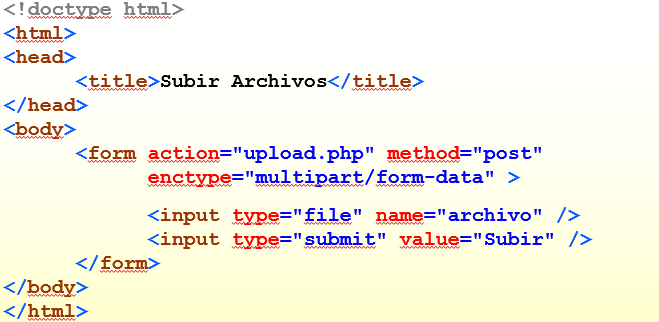
El par de nombres / valores es enviado en el cuerpo del mensaje HTTP. Las peticiones POST no se almacenan en el caché. No permanecen en el historial del navegador. No pueden ser marcadas. No poseen restricciones de longitud de datos.

**MANEJO DE FORMULARIOS:**

* Tanto GET como POST crean un *array asociativo*.
* Dicho array contiene pares de clave-valor, dónde las claves son los nombres (atributo ***name***) de los controles del formulario y los valores son la entrada de datos del usuario.
* PHP utiliza las ***super globales*** $\_GET, $\_POST y $\_REQUEST para recolectar datos provenientes de un Form.
* $\_GET es un array pasado por GET.
* $\_POST es un array pasado por POST.
* $\_REQUEST es un array asociativo que contiene $\_GET, $\_POST y $\_COOKIE.

**CLASE 04 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Subir archivos al servidor:** Para poder subir archivos, es necesario crear un formulario HTML, que permita seleccionarlos.



**Algunas reglas a seguir para el formulario HTML:**

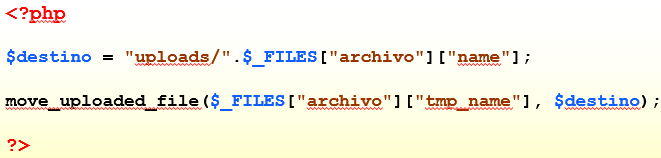
\* El método del form debe ser POST.

\* El form necesita del atributo **enctype**, este especifica el contenido / tipo a usarse cuando se “submitea” el form.

\* El input de tipo FILE permite mostrar una ventana modal para navegar en busca del archivo a ser subido.

SIN LOS REQUERIMIENTOS INDICADOS, LA SUBIDA DEL ARCHIVO FALLARÁ.

Del lado del servidor, tenemos que manipular el archivo recibido en **$\_FILES**. Utilizando funciones de PHP, deberemos **mover** el archivo subido desde su **ubicación** **temporal** a la **ubicación** **definitiva** **dentro** **del** **servidor**.

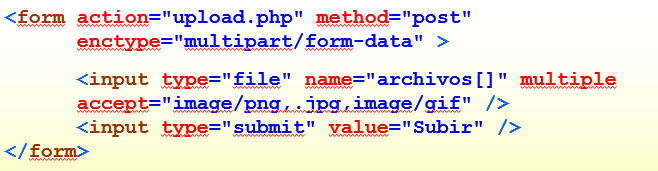


**$FILES:** Es una super global que existe a partir de PHP (4.1.0).

Es un array asociativo de elementos cargados al script actual a través del método POST.

* + **[name]** => nombre del archivo (con su extensión).
  + **[type]** => tipo del archivo (dado por el navegador).
  + **[tmp\_name]** => carpeta temporal dónde se guardará el archivo subido.
  + **[error]** => código de error (si es 0, no hubo errores).
  + **[size]** => tamaño del archivo, medido en bytes.

**ATRIBUTOS:** El nuevo atributo booleano **multiple** (de HTML5) permite que los usuarios seleccionen varios archivos para ser subidos al servidor. El atributo **accept** permite filtrar (en el cliente) los tipos de archivos que se permitirán subir al servidor.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**VARIABLES DE SESION:**

**Generalidades.**

Las variables de sesión son una forma de guardar información de un usuario particular. La información no es almacenada en el cliente. Dado que HTTP no mantiene estado entre páginas, la utilización de variables de sesión permite mantener información acerca de un solo usuario. Por defecto las variables de sesión duran **hasta** que el usuario **cierra** **el** **navegador**.

**Abrir una sesión:** Se inicia con **session**\_**start**(). Dicha función, debe estar declarada en cada script al que deseamos utilizar con variables de sesión. Las variables de sesión se establecen con la super global de PHP: **$\_SESSION**.

**session\_start();**

**$\_SESSION**[“CLAVE”] = “VALOR”;

La mayoría de las sesiones establecen una clave parecida a **“*765487cf34ert8dede5a562e4f3a7e12”.*** Cuando una sesión se abre en otra página, se examina el equipo para obtener una clave de usuario. Si hay una coincidencia, se accede a esa sesión, sino se inicia una nueva sesión.

**Cerrar una sesión:** Para eliminar todas las variables de sesión globales y destruir la sesión, hay que usar:

**session\_unset();** // remueve todas las variables de sesión

**session\_destroy();** // destruye la sesión

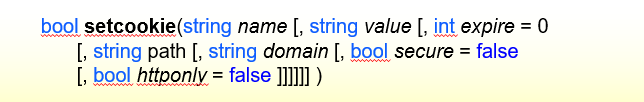
**COOKIES:**

**Generalidades:**

* Una cookie se utiliza a menudo para identificar a un usuario.
* Una cookie es un pequeño archivo que el servidor guarda en el cliente.
* Cada vez que el mismo equipo solicita una página con un navegador, se enviará la cookie también.
* Con PHP, se puede tanto crear como recuperar valores de cookies.

**Establecer una cookie:**

* ***setcookie()*** define una cookie para ser enviada junto con el resto de las cabeceras de HTTP.
* Al igual que otras cabeceras, las cookies deben ser enviadas antes de que el script genere ninguna salida (es una restricción del protocolo).
* Esto implica que las llamadas a esta función se coloquen antes de que se genere cualquier salida, incluyendo las etiquetas **<html>** y **<head>** al igual que cualquier espacio en blanco.
* Una vez que han sido enviadas las cookies, se puede acceder a ellas en la próxima carga de la página gracias a los Arrays **$\_COOKIE**.



**name:** Indica el nombre de la cookie.

**value:** Establece el valor de la cookie. Este valor se guarda en el cliente.

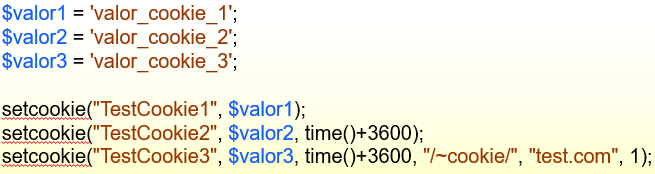
**expire:** indica el tiempo en el que expira la cookie. Es una fecha Unix por tanto está expresada en números de segundos a partir de la presente época.

**path:** Indica la ruta dentro del servidor en la que la cookie estará disponible.

**domain:** El dominio para el cual la cookie está disponible.

**secure:** Establece si la cookie sólo debiera transmitirse por una conexión segura HTTPS desde el cliente. Cuando se configura como TRUE, la cookie sólo se creará si es que existe una conexión segura.

**httponly:** Cuando es TRUE la cookie será accesible sólo a través del protocolo HTTP. Esto significa que la cookie no será accesible por lenguajes de scripting, como JavaScript.



**Borrar una cookie:** Para borrar una cookie se debe asegurar que la fecha de expiración ya ha pasado, de este modo se detonará el mecanismo de eliminación del navegador.

**setcookie**("TestCookie2", " ", time()-3600);

**setcookie**("TestCookie3", " ", time()-3600, "/~cookie/", "test.com", 1);

**CLASE 06 - PDO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Introducción a PDO:**

PDO define una interfaz ligera para poder acceder a bases de datos en PHP. PDO proporciona una capa de abstracción de acceso a datos, lo que significa que independientemente de la base de datos, se implementan las mismas funciones.

**Conexiones:**

Las conexiones se establecen creando instancias de la clase base PDO. No importa el controlador que se utilice, siempre se usará el nombre de la clase PDO. El constructor acepta parámetros para especificar el origen, el nombre de usuario, y la contraseña.

$conStr = **'mysql**:**host** = localhost; **dbname** = pruebaDB';$**pdo** = new **PDO**($**conStr**, $**user**, $**pass**);

**En caso que exista una exception:**

**try {**

**$conStr = 'mysql:host = localhost; dbname = pruebaDB';  
 $pdo = new PDO($conStr, $user, $pass);**

**}**

**catch(PDOException $e){**

**echo ''Error: '' .$e->getMessage() . ''<br/> '';**

**}**

Una vez realizada con exito una conexion a la base de datos, será devuelta una instancia de la clase PDO al script.

La conexion permanecerá active durnate el tiempo de vida del objeto PDO.

Para cerrar la conexion, es necesario destruir el objeto asegurandose que tofdas las referencias a él sean eliminadas. (Asignando NULL a la variable que obtiene el objeto). Si no se realiza explicitamente, se cerrará automaticamente.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**fetchAll()**

Devuelve un array que contiene todas las filas de un conjunto de resultados. El parámetro fetch\_style determina el estilo.

**mixed fetchAll( [ $fetch\_style] )**

**fetchObject()**

Obtiene la siguiente fila y la devuelve como un objeto. Esta función es una alternativa para PDOStatement::fetch() con el estilo PDO::FETCH\_CLASS o PDO::FETCH\_OBJ.

**mixed fetchObject( [ $className ,[ $args] ] )**

* **$className: Nombre de la clase creada.**
* **$args: Los elementos de este array son pasados al constructor*.***

**\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SENTENCIAS PREPARADAS:**

Muchas de las bases de datos más maduras admiten las sentencias preparadas. Estas pueden definirse como tipo de plantillas compiladas para SQL. Ofrecen 2 grandes beneficios:

(1) La consulta solo necesita sere analizada una vez. Pero puede ser ejecutada muchas veces con los mismos o diferentes parámetros. (Las sentencias utilizan menos recursos y se ejecutan más rápidamente).

(2) Los parámetros para las sentencias preparadas no necesitan estar entrecomillados, el controlador automáticamente se va a encargar de eso.

Las sentencias preparadas son tan utiles que son la única característica que PDO emulará para los controladores que no las soporten. Esto asegura que una aplicación emplee el mismo paradigma, sin importar las capacidades de la base de datos.

Las declaraciones preparadas básicamente funcionan así:

**Prepare()** Una plantilla de declaración de SQL se crea y se envía a la base de datos. Retorna PDOStatement. La base de datos analiza, compila y realiza la optimización de consultas en la plantilla y almacena el resultado sin ejecutarlo.

**Execute()** En un momento posterior, la aplicación enlaza (bindea) los valores a los parámetros y a la base de datos ejecuta la instrucción.

SENTENCIA PREPARADA SIN PARAMETROS.

**$sentencia = $pdo->prepare(‘SELECT \* FROM tabla’);  
$sentencia->execute();**

SENTENCIA PREPARADA CON PARAMETROS.

**$sentencia = $pdo->prepare('SELECT \* FROM tabla WHERE ID = :id');  
$sentencia->execute(array(':id' => 3));**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PDOStatement**

Representa una sentencia preparada y, después de la ejecución de la instrucción, un conjunto de resultados asociado. Posee métodos para vincular (bindear) valores a parámetros. Posee métodos para obtener los valores de un conjunto de resultados.

**METODOS PARA VINCULAR:**

**BINDPARAM()** Vincula una variable de PHP a un parámetro de sustitución con nombre o de signo de interrogación correspondiente de la sentencia SQL que fue usada para preparar la sentencia.

**bool bindParam( $param, $variable [, $tipo [, $length ]] )**

* **$param:** Identificador del parámetro.
* **$variable:** Nombre de la variable de PHP a vincular al parámetro de la sentencia SQL.
* **$tipo:** Tipo de dato explícito para el parámetro, usando las constantes PDO::PARAM\_\*.
* **$length**: Longitud del tipo de datos.

**Parámetros nombrados:** $sentencia = $pdo->prepare('SELECT \* FROM tabla WHERE ID = **:id**');

$sentencia->bindParam(**':id'**, $var, PDO::PARAM\_INT); $sentencia->execute();

**Parámetros posicionales:** $sentencia = $pdo->prepare('SELECT \* FROM tabla WHERE ID = **?**');

$sentencia->bindParam(**1**, $var, PDO::PARAM\_INT); $sentencia->execute();

**BINDVALUE()** Vincula un valor al parametro de sustitucion con nombre o de signo de interrogacion correspondiente de la sentencia SQL que fue usada para preparar la sentencia.

**bool bindValue( $param, $valor [, $tipo] )**

* **$param:** Identificador del parámetro.
* **$valor:** Valor a vincular al parámetro de la sentencia SQL.
* **$tipo:** Tipo de dato explícito para el parámetro, usando las constantes PDO::PARAM\_\*.

**Parametros nombrados:**  $sentencia = $pdo->prepare('SELECT \* FROM tabla WHERE ID = **:id**');

$sentencia->bindValue(**':id'**, 3, PDO::PARAM\_INT); $sentencia->execute();

**Parametros posicionales:** $sentencia = $pdo->prepare('SELECT \* FROM tabla WHERE ID = **?**');

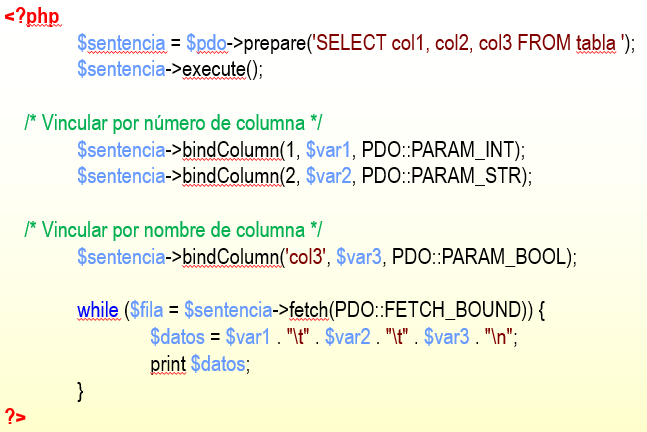
$sentencia->bindValue(**1**, $variable, PDO::PARAM\_INT); $sentencia->execute();

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BINDCOLUMN()** Vincula una columna a una variable de PHP. Cada llamada a ***PDOStatement::fetch()***  o a ***PDOStatement::fetchAll()*** actualizará todas las variables que estén vinculadas a columnas.

**bool bindColumn( $column, &$variable [, $tipo [, $maxLen ]] )**

* **$column**: Número (base 1) o nombre de la columna del conjunto de resultados.
* **$variable:** Nombre de la variable de PHP a la que vincular la columna.
* **$tipo:** Tipo de dato explícito para el parámetro, usando las constantes PDO::PARAM\_\*.
* **$maxLen:** Longitud máxima sugerida para la pre asignación.



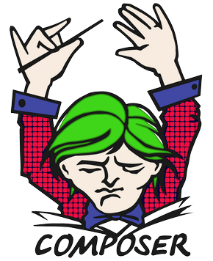
**METODOS PARA OBTENER VALORES:**

**FETCH()**  Obtiene una fila de un conjunto de resultados asociado al objeto *PDOStatement*. El parámetro **fetch**\_**style** determina cómo PDO devuelve la fila**.**

**mixed fetch( [ $fetch\_style] )**

* **$fetch\_style**: Este valor debe ser una de las constantes *PDO::FETCH\_\*.*
* El valor de retorno de esta función en caso de éxito depende del tipo de obtención. En todos los casos, se devuelve **FALSE** en caso de error.
* ***PDO::FETCH\_ASSOC*:** devuelve un array indexado por los nombres de las columnas del conjunto de resultados.
* ***PDO::FETCH\_NUM*:** devuelve un array indexado por el número de columna tal como fue devuelto en el conjunto de resultados, comenzando por la columna 0.
* ***PDO::FETCH\_BOTH*** (predeterminado): devuelve un array indexado tanto por nombre de columna, como numéricamente con índice de base 0 tal como fue devuelto en el conjunto de resultados.
* ***PDO::FETCH\_OBJ*:** devuelve un objeto anónimo con nombres de propiedades que se corresponden a los nombres de las columnas devueltas en el conjunto de resultados.
* ***PDO::FETCH\_CLASS*:** devuelve una nueva instancia de la clase solicitada, haciendo corresponder las columnas del conjunto de resultados con los nombres de las propiedades de la clase.
* ***PDO::FETCH\_BOUND*:** devuelve TRUE y asigna los valores de las columnas del conjunto de resultados a las variables de PHP a las que fueron vinculadas con el método PDOStatement::bindColumn().

**CLASE 07 – COMPOSER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Generalidades:**  Es una herramienta para la gestión de dependencias en PHP. Permite declarar las bibliotecas de las que depende tu proyecto y las va a instalare y actualizar.

**¿Por qué utilizarlo?**

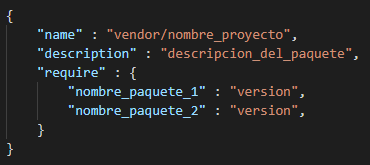
\* Cuando se comienza un proyecto en PHP con cierta complejidad, no sirve solo con las funciones nativas de PHP. Generalmente se necesitan librerias de 3ros.

\* Ya sea un framework, sistemas de envio, etc.

\* Los gestores de paquetes ayudan a resumir las tareas de descarga.

\* Los gestores de paquetes ya existian en otros lenguajes de programacion cono npm en NodeJS.

**¿Cómo funciona?** Para beneficiarnos del workflow que nos propone Composer, simplemente tenemos que escribir un archivo de configuracion. El archivo es un simple JSON , en el que se indican cosas cómo: El autor del proyecto, las dependencias, etc. El archivo debe tener un nombre especifico (muy importante) -> **composer.json**

**EJEMPLO DE COMPOSER.**

**UNA VEZ FINALIZADA LAS DEPENDENCIAS, SE DEBE INSTALAR CON:**

****

Una vez finalizado el proceso en la consola de comandos, se podrá encontrar en la carpeta raíz del proyecto, en un directorio llamado “vendor”, donde estarán las librerías declaradas.

Solo queda hacer los include para que estén disponibles las aplicaciones y para ello también nos ayuda Composer.

Simplemente tendremos que hacer un único include o require en nuestro código, y todas las librerías estarán disponibles para utilizarse.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PACKAGIST:**

Se trata del repositorio de paquetes que son instalables mediante Composer. En cada paquete hay información y el código necesario para declarar la dependencia en el JSON de Composer -> <https://packagist.org/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INSTALACION:** **WINDOWS:** Instalador. Siguiente, siguiente, finalizar. **LINUX:** **curl -sS** [**https://getcomposer.org/installer**](https://getcomposer.org/installer) **| php**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ESQUEMA COMPOSER.JSON:**

**authors:** Es un array con los autores del proyecto. A cada uno de los elementos se le puede indicar: name, email, homepage, role.

**require:** Es un objeto con una serie de pares clave/valor que definen cada una de las dependencias que Composer debe instalar para nuestro proyecto. En la clave debemos de indicar el nombre del paquete que depende (que se obtiene del sitio de Packagist**)** y como valor indicamos la versión que deseamos que esté instalada, o el rango de versiones.

**Versión exacta:** indica que se debe instalar una versión exacta, y solo esa.

**Rango de versiones:** permite indicar versiones que sean mayor que una determinada, menor o que esté entre una versión y otra.

**Comodín:** Permite decir cualquier versión de una release mayor. Ejemplo "4.\*"

**Virgulilla (~):** permite indicar la versión de una manera diferente, "próxima versión significativa". Por ejemplo, "~2.2" siempre te dejará la versión mayor o igual a la 2.2 y menor que la 3.0.