EJERCICIOS PHP
RELACIÓN IV - SESIONES, COOKIES, POO Y JSON (Segunda versión)

1- Vamos a crear una aplicación php muy básica en dos archivos:

- **login.php:** una página html con un formulario de acceso que me pida id de usuario y contraseña (ambos valores se validarán con js)

- **proceso.php:** este archivo simplemente creará una sesión, comprobará que el id de usuario y la contraseña son los que deben ser invocando a una función php con valores locales para ambos datos (lo comprueba mediante dos simples if y devuelve un booleano) y establece una cookie con el nombre y una variable de sesión:

  ```
  setcookie("id_usuario", $nombre_usuario, time() + 30, "/");
  $_SESSION["id_usuario"]=$_POST[`nombre_usuario’];
  ```

  A continuación, mediante dos echo, muestra el nombre a través de cookie y el nombre a través de variable de sesión.

  Refresca la página cada 5 segundos…Ten en cuenta que mientras no se recargue la página, la cookie no está activa, y que dejará de estarlo a los 30 segundos

2- Hagamos un sencillo programa que demuestre de forma práctica el uso de las variables de sesión y su mantenimiento. Para ello, mostraremos dos variables diferentes: num1 y num1, que inicialmente, valdrán 0 ambas, y un campo select de un formulario, en el que podremos elegir cuatro opciones:

- aumentar a
- disminuir a
- aumentar b
- disminuir b

al elegir una de estas opciones, el programa se llamará a sí mismo e incrementará o disminuirá el valor de la variable correspondiente. Si no se desplegase el campo select, por defecto, aumentará la variable a (no hará falta que valides aquí el haberlo desplegado o no.

Cierra la pestaña y vuelve a abrirla ¿Qué ocurre si quieres empezar de nuevo de cero?¿Por qué? Prueba en otro navegador. Piensa en las implicaciones de esto.

Incluye tres nuevas opciones en el desplegable que te permitan:

- resetear a
- resetear b
- destruir la sesión, y refrescar la página (consulta este artículo, especialmente el apartado 1: Actualizar una página usando PHP - GeeksforGeeks)

3- Vamos a hacer un juego en PHP. Al comienzo, el servidor debe "pensar" un número entre 1 y 100. El usuario introduce mediante un formulario, el número que cree que ha "pensado" el servidor. Si ha acertado, recibe la enhorabuena, si no, una pista: "te has pasado" o "te has quedado corto", y ofrecerá un nuevo intento al usuario.

Deberás hacerlo todo (formulario y procesamiento) en un mismo archivo. En modo depuración, utilizaremos el método get para pasar los valores, y usaremos para mantener el número entre varias llamadas, el paso de campos de formulario de tipo hidden, que es un recurso que todavía no hemos utilizado y se revelará de gran utilidad en un futuro. No utilizaremos, por tanto, sesiones.

4- Haremos una segunda versión del programa anterior, pero esta vez, en vez de utilizar el recurso de campos "hidden", usaremos el establecimiento de sesión y de variables de sesión. ¿Cuál crees que es la forma más acertada de haber resuelto este problema? ¿Por qué?

5- Crea un programa en PHP que defina la clase **Restaurante**, con los siguientes atributos:

- nombre
- tipoCocina
- ratings (es un array de valoraciones numéricas enteras entre 1 y 5)

Define los métodos:

- constructor (solo incluirá como parámetros el nombre y el tipoCocina e inicializará ratings a vacío)
- destructor
- toString (que permita visualizarla de una forma estructurada y estética) OJO: __toString es un "método mágico" (los veremos más adelante)
- obtener el número de ratings (el número de votos)
- añadir un rating
- añadir otros ratings (más de uno)
- calcular el rating medio

Y pruébalos para comprobar que todo funciona correctamente

6- Modifica la clase anterior para que:

- utilice promoción de propiedades
- todos los atributos sean privados y solo se pueda acceder a ellos mediante getters y setter. Añade estas funciones al código.
  OJO: también existen __get() y __set() mágicos, pero no los usaremos aún
- tenga un **atributo static** a la clase: numeroRest, privado, que se incremente con cada nuevo objeto y se pueda acceder a él mediante un método público de clase totalRests()

7- Vamos a crear una clase BanderaFranjas con los siguientes atributos:

- horizontal/vertical según sea la orientación de las franjas
- la lista de franjas de colores que la define
- nombre de país u organización (por defecto, "sin adscripción")

Vamos a crear para ella:
- un método constructor
- un método destructor
- un método que muestre la bandera por pantalla con el nombre de los colores en franjas
- un método que compare dos banderas y diga si son idénticas.
- un método que compare dos banderas y diga si tiene las mismas franjas en diferente orientación
- un método que invierta el orden de los colores
- un método que invierta la orientación de las franjas

Prueba que todo funcione correctamente

8- Define una clase **CuentaBancaria** con los siguientes atributos privados:
- número de cuenta
- nombre del titular
- saldo (inicial, 0 euros)
- número de operaciones (inicial, 0)

Define también las siguientes operaciones:
- constructor
- destructor
- toString
- depositar dinero
- extraer dinero
- transferir dinero (de esta cuenta a otra cuenta)

9- En el ejercicio anterior, no estamos teniendo en cuenta si está permitido que una cuenta tenga el saldo negativo (cuenta de crédito) o no (cuenta de débito). En realidad, todo es prácticamente igual, salvo que en la de crédito, hay un límite máximo para el saldo negativo, y en la de débito, no se puede extraer dinero si el saldo no es suficiente.

Reestructura el ejercicio y define la clase CuentaBancaria como abstracta e instancia dos clases hijas **CuentaDebito** y **CuentaCredito** que tengan todo esto en cuenta en su operatoria.

Prueba a crear dos cuentas, a hacer ingresos, extracciones y transferencias, y comprueba que todo funciona correctamente.

10- Vamos a experimentar que es una interfaz y cómo se construyen en PHP:

- define una interfaz **Encendible** como aquella que tiene únicamente dos métodos abstractos: encender() y apagar()

- define una clase **Bombilla** que implemente esa interfaz y contenga además otros dos atributos y dos métodos:
  - tipoBombilla : ya que la bombilla puede ser incandescente, led, halógena o fluorescente
  - lúmenes: la cantidad de luz que pueden generar en Watios
  - encendida (true o false, inicialmente false)
  - método constructor con dos parámetros para tipo y lúmenes
  - método destructor para eliminar la bombilla
  - tanto encender como apagar serán sencillos, únicamente pondrán el estado sin comprobación previa y emitirán un mensaje

- define una clase **Motocicleta** que implemente también esta interfaz y contenga también los siguientes atributos y métodos adicionales:
  - gasolina
  - batería
  - matrícula
  - estado
  - método constructor que inicialmente pone la gasolina a 0 y la energía a 2 por defecto, siempre pone el estado a apagado, e inicializa la matrícula
  - método cargarGasolina, que suma litros al depósito
  - método encender: se comprobará que no esté ya encendida, que haya batería y que haya gasolina (si alguna de estas condiciones no se da, emite un mensaje al respecto), emitiendo un mensaje de confirmación

    - método apagar: debe comprobar que esté encendido el motor y emite el mensaje correspondiente

- si todo es correcto, podremos definir y ejecutar el siguiente código:

```
function enciende_algo (Encendible $algo) {
        $algo->encender();
}
$miBombilla = new Bombilla("led",12);
$miMoto = new Motocicleta("3873 NXB");

enciende_algo($miBombilla);
enciende_algo($miMoto);
```

11- Vamos ahora a experimentar con stdClass, que es la clase genérica vacía en PHP que se utiliza para crear objetos dinámicos, sin una estructura de clase formal definida.

Vamos a crear un objeto nuevo llamado **móduloDWES** perteneciente a esta clase, para, a continuación, incluir en él los siguientes atributos:

- módulo: un string, por ejemplo "Desarrollo Web en Entorno Servidor"
- acrónimo: otro string, por ejemplo "DWES"
- curso: numérico entero, en este ejemplo, 2
- descripción: una descripción de contenidos esquemática
- teacher: nombre de quien lo imparte

Una de las propiedades más interesantes de los objetos stdClass es que se pueden convertir en arrays mediante casting con (array) y viceversa: un array se puede convertir en Object con (object). Experiméntalo y muéstralo en cada etapa de su transformación del derecho y del revés.

12- Tanto objetos como arrays adolecen de un problema importante cuando manejan una cantidad de datos significativa que además, queremos conservar: son estructuras de datos residentes en memoria principal, y por tanto, volátiles, perecen cuando se carga otro módulo php que no sea el actual.

Para poder conservar su información, hay que buscar la forma de convertirlas en una cadena. Existen varias opciones, una de ellas es convertirlas en string y almacenarlas en una cookie o, de forma más segura, en una variable de sesión.

Para ello, podemos utilizar las funciones **serialize** y **unserialize**. Consulta la documentación oficial, y añade al ejercicio 11 una conversión de este tipo para el array y para el objeto. Muestra por pantalla el resultado de serialize y unserialize en ambos casos.

13- Otra forma de convertir objetos y arrays asociativos en cadenas, es utilizar el formato JSON.

Declara un array asociativo de socios de un club deportivo, con al menos 3 socios, de los cuales almacenaremos nombre, apellidos y edad.

Convierte el array en json con **json_encode** (Consulta: PHP: json_encode - Manual) y muestra su contenido antes y después de esta conversión.

Ahora, convierte el objeto json con **json_decode** (consulta documentación oficial: PHP: json_decode - Manual) de vuelta en array asociativo, y en objeto stdClass

14- Vamos a imaginar que estemos realizando adquisiciones en una tienda online.
Crea un array asociativo con códigos, nombres de artículos, y número de unidades de esos artículos, al menos 3. No nos ocuparemos de como han llegado esos datos (en realidad, sería a través de la interacción del usuario con el front). A continuación, codifica el array como json utilizando la función **json_encode** y muestra el resultado de esa conversión por pantalla.

Almacena el carrito en una cookie sin tiempo de finalización.

La página contendrá un botón, que al pulsarlo, funcionará como un vínculo a otra página.

Ahora, en esta segunda página, captura la cookie, haz la conversión inversa con **json_decode** y muestra su contenido. Prueba a convertirlo en array asociativo y también en objetos de la clase stdClass.

Imagina (no es necesario que lo construyas) que hay una interfaz de usuario que permite añadir al carrito los artículos, y otra que es capaz de acceder con el contenido recibido a través de la cookie a una base de datos, para gestionar la compra y (después de utilizar una pasarela de pago), la facturación y el envío del pedido. Pero eso no lo haremos, al menos por ahora

15- PHP es un lenguaje, que en un principio tenía un tipado débil. Pero a partir de la versión 7, dejó de ser así. No obstante, hasta el momento,

hemos estado creando funciones en las que no hemos indicado ni el tipo de los parámetros, ni el tipo de los valores devueltos.

Esto hace nuestro código "compatible hacia atrás", pero es desaconsejable cuando estamos trabajando con clases, objetos y toda su complejidad.

Revisa todos los métodos definidos en esta relación en los ejercicios del 5 al 10, y utiliza tipos para parámetros y valores devueltos.

Por otra parte, un tipado estricto va acompañado de problemas de manejo del valor null, por lo que el lenguaje también provee de la posibilidad de manejar este valor: **Null Safety,** que nos resultará de vital importancia cuando trabajemos con objetos que son recursos

Consulta estos artículo para más aclaraciones Funciones con Tipado Definido en PHP | Blog Coders Free y

https://www.slingacademy.com/article/nullable-optional-types-in-php-a-practical-guide-5-examples/

16- Vamos a ver ahora dos recursos que se utilizan cuando el número de archivos y librerías de un proyecto puede hacerse grande e inmanejable: namespaces y require.

Los **namespaces** en PHP son un recurso que se utiliza para organizar el código y evitar conflictos de nombres entre clases, funciones o constantes. Cuando el código está dividido en múltiples archivos y/o utiliza diversas librerías, se pueden producir conflictos con los nombres.

require incluye y ejecuta el contenido de un archivo en el script actual. Si el archivo no se encuentra, produce un error fatal que detiene la ejecución del script. Se usa para incluir archivos esenciales como configuraciones críticas o dependencias de los que el script depende para funcionar.

Consulta este artículo e implementa el ejemplo que viene en él, y este otro para conocer los parecidos y las diferencias entre include, require, include_once y require_once: La diferencia entre require, include, require_once, include_once - programador clic

17- Los **traits** son una forma de "simular" la herencia múltiple en PHP ya que no existe como tal. Es una forma de modularizar y re-aprovechar código. En el siguiente artículo, se presenta un ejemplo de declaración y uso de traits. Implementa y practica el ejemplo que contiene: https://bcube.bitban.com/blog/traits-de-php

OJO: Ahora mismo, require, y traits no nos son especialmente útiles, pero cuando veamos Laravel, los rescataremos y utilizaremos a fondo.