Unknown Title

EJERCICIO 3. LISTA DE TAREAS CON COOKIES

Realiza un formulario para realizar un to-do list o lista de tareas. El formulario debe mostrarnos un cuadro de texto (será la descripción de una tarea) y el botón "Añadir".

El script php almacenará dicha tarea internamente dentro de un array en la tabla global cookies del navegador y de esta manera podremos mantener todas las listas de tareas entre las distintas páginas. Para ello almacenar cada nueva tarea. se debe usar la función array_push.

Al pulsar añadir la tarea se añadirá a una lista que contienen las tareas.

Una vez añadida, se mostrará la lista completa de tareas.

Para resolver este ejercicio, utilizaremos PHP para gestionar una cookie que almacene un array de tareas. Dado que las cookies solo pueden guardar texto, utilizaremos las funciones `serialize()` y

`unserialize()` (o `json_encode` / `json_decode`) para convertir el array en una cadena de texto y viceversa.

Aquí tienes una propuesta de solución estructurada y funcional:

Solución: Lista de Tareas con PHP y Cookies

```
<?php
// 1. Nombre de la cookie
$cookie_name = "lista_tareas";

// 2. Recuperar la lista actual si existe
if (isset($_COOKIE[$cookie_name])) {
    // Deserializamos el contenido para convertir la cadena de texto de
nuevo en un array
    $tareas = unserialize($_COOKIE[$cookie_name]);
} else {
    // Si no existe, inicializamos un array vacío
    $tareas = array();
}

// 3. Lógica para añadir una nueva tarea
if (isset($_POST['añadir']) && !empty($_POST['tarea'])) {
    $nueva_tarea = htmlspecialchars($_POST['tarea']); // Limpieza básica de
datos

    // Usamos array_push para añadir la nueva tarea al array existente
    array_push($tareas, $nueva_tarea);

    // Guardamos el array actualizado en la cookie (duración: 1 día / 86400
seg)
    // Es necesario serializarlo para que se guarde como string
    setcookie($cookie_name, serialize($tareas), time() + 86400, "/");

    // Recargamos la página para que los cambios se reflejen (opcional,
pero recomendado)
    header("Location: " . $_SERVER['PHP_SELF']);
    exit;
}

// 4. Lógica para borrar la lista (Opcional, para facilitar las pruebas)
if (isset($_POST['limpiar'])) {
    setcookie($cookie_name, "", time() - 3600, "/");
    header("Location: " . $_SERVER['PHP_SELF']);
```

```
    exit;
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>To-Do List con Cookies</title>
</head>
<body>

    <h2>Mi Lista de Tareas</h2>

    <form method="post" action="">
        <input type="text" name="tarea" placeholder="Escribe una tarea..."
required>
        <button type="submit" name="añadir">Añadir</button>
        <button type="submit" name="limpiar" style="color: red;">Limpiar
Todo</button>
    </form>

    <hr>

    <h3>Tareas Pendientes:</h3>
    <ul>
        <?php
        if (!empty($tareas)) {
            foreach ($tareas as $item) {
                echo "<li>" . $item . "</li>";
            }
        } else {
            echo "<li>No hay tareas pendientes.</li>";
        }
        ?>
    </ul>

</body>
</html>
```

Puntos clave del código:

- **`serialize()` / `unserialize()`**: PHP no permite guardar un array directamente en una cookie. Al serializarlo, convertimos la estructura de datos en un formato de texto que el navegador puede

almacenar.

- **`array_push()`**: Como solicitaste, se utiliza para insertar el valor enviado por el formulario al final del array recuperado de la cookie.
- **Persistencia**: Al establecer el tiempo de vida (`time() + 86400`), las tareas seguirán ahí aunque cierres el navegador, hasta que la cookie expire o sea eliminada.
- **`setcookie()`**: Es importante recordar que esta función debe llamarse **antes** de que el script envíe cualquier salida HTML al navegador.

¿Te gustaría que añadiera una opción para poder borrar tareas individuales de la lista?

- Consulta realizada correctamente

Entiendo que los archivos que has compartido utilizan actualmente **Sesiones** (`$_SESSION`) para gestionar la lista de tareas. Sin embargo, el ejercicio solicita específicamente el uso de **Cookies** (`$_COOKIE`).

Las cookies tienen una diferencia fundamental: los datos se guardan en el navegador del usuario y, al ser texto plano, debemos **serializar** el array para guardarlo y **deserializarlo** para leerlo.

Aquí tienes la adaptación de tus archivos para cumplir con el requisito de las cookies:

1. `arrays.php`

Este archivo ahora leerá de `$_COOKIE` en lugar de `$_SESSION`.

```
<?php
$cookie_name = "mis_tareas";

// Si la cookie existe, la deserializamos para obtener el array.
// Si no, inicializamos un array vacío.
if (isset($_COOKIE[$cookie_name])) {
    $tareas = unserialize($_COOKIE[$cookie_name]);
} else {
    $tareas = array();
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
```

```
    <title>Ejercicio 3 - Cookies</title>
</head>
<body>
    <h1>Ejercicio 3 - Lista con Cookies</h1>
    <form action="gestion_cookies.php" method="POST">
        <p>
            <label for="tarea">Nueva Tarea:</label>
            <input type="text" id="tarea" name="tarea" placeholder="Escribe
tu tarea" required>
            <input type="submit" value="Añadir" />
        </p>
    </form>

    <form action="gestion_cookies.php" method="POST">
        <h2>Tareas pendientes</h2>
        <?php
        if (!empty($tareas)) {
            foreach ($tareas as $indice => $tarea) {
                echo "<p>";
                echo "<input type='checkbox' name='borrar[]'
value='{$indice}'> ";
                echo htmlspecialchars($tarea);
                echo "</p>";
            }
            echo "<input type='submit' value='Borrar Tareas Marcadas'/>";
        } else {
            echo "<p>No hay tareas en la lista.</p>";
        }
        ?>
    </form>

    <br>
    <a href="logout.php">Borrar todas las Cookies (Reset)</a>
</body>
</html>
```

---

2. `gestion_cookies.php`

Aquí procesamos la lógica. Usamos `setcookie` para persistir los cambios.

```
<?php
$cookie_name = "mis_tareas";

// Recuperamos el array actual de la cookie
```

```
if (isset($_COOKIE[$cookie_name])) {
    $tareas = unserialize($_COOKIE[$cookie_name]);
} else {
    $tareas = array();
}

// LÓGICA PARA AÑADIR
if (isset($_POST['tarea']) && trim($_POST['tarea']) != "") {
    // Usamos array_push como pide el ejercicio
    array_push($tareas, $_POST['tarea']);
}
// LÓGICA PARA BORRAR
elseif (isset($_POST['borrar']) && is_array($_POST['borrar'])) {
    $indices_a_borrar = $_POST['borrar'];
    rsort($indices_a_borrar); // Orden descendente para no desordenar el
índice al borrar

    foreach ($indices_a_borrar as $indice) {
        array_splice($tareas, $indice, 1);
    }
}

// GUARDAR CAMBIOS: Serializamos el array y actualizamos la cookie (expira
en 24h)
setcookie($cookie_name, serialize($tareas), time() + 86400, "/");

// Redirigir de vuelta
header("Location: arrays.php");
exit();
?>
```

---

3. `logout.php` (Reset de Cookies)

Para "cerrar sesión" con cookies, debemos indicar al navegador que la elimine (poniendo una fecha de expiración pasada).

```
<?php
    // Para borrar una cookie, se pone una fecha de expiración en el pasado
    setcookie("mis_tareas", "", time() - 3600, "/");
    header("Location: arrays.php");
    exit();
?>
```

Cambios principales realizados:

1. **Sustitución de Sesión**: Se eliminó `session_start()` y las referencias a `$_SESSION`.
2. **Serialización**: Como las cookies no admiten arrays nativos, usamos `serialize()` para convertir el array en un "string" almacenable y `unserialize()` para recuperarlo.
3. **Persistencia**: Con `setcookie()`, los datos se mantienen incluso si cierras el navegador, a menos que uses el enlace de "Cerrar Sesión".

¿Te gustaría que te explicara más a fondo cómo funciona el proceso de `serialize()` para enviar datos por el navegador?

Que hace $cookie_name = "mis_tareas";

En el código PHP que hemos desarrollado, la línea `$cookie_name = "mis_tareas";` cumple una función organizativa y de identificación fundamental.

Aquí te explico qué hace exactamente:

1. Definición de un identificador (Nombre)

Esta línea crea una **variable** que almacena una cadena de texto. Este texto será el "nombre" que el navegador utilizará para guardar tu lista de tareas. Imagina que es la etiqueta de una caja donde guardas los datos; sin esa etiqueta, el navegador no sabría qué información entregarte cuando la pidas.

2. Facilita el mantenimiento

Al guardar el nombre en una variable en lugar de escribir `"mis_tareas"` directamente en cada función, haces que el código sea más fácil de modificar.

- Si mañana quieres cambiar el nombre de la cookie a `"lista_usuario_1"`, solo tienes que cambiarlo en esa línea y se actualizará automáticamente en todo el script.

3. Conexión con las funciones de Cookies

Esta variable se utiliza como el primer argumento en las funciones clave de PHP:

- **Para leer:** `$_COOKIE[$cookie_name]` busca en el navegador si existe algo guardado bajo ese nombre.
- **Para crear/modificar:** `setcookie($cookie_name, ...)` le ordena al navegador: "Guarda estos datos bajo el nombre que indica esta variable".
- **Para borrar:** `setcookie($cookie_name, "", time() - 3600, ...)` le dice al navegador que elimine la "caja" con ese nombre específico.

¿Te gustaría que veamos cómo añadir una fecha de expiración para que la lista dure más de un día?

explicame lo de serialize y unserialize

Como hemos visto en el ejercicio, las cookies tienen una limitación importante: **solo pueden almacenar texto (strings)**. No puedes guardar un array directamente en ellas. Aquí es donde entran en juego `serialize()` y `unserialize()`.

Imagina que estas funciones son como una empresa de mudanzas que necesita enviar un mueble (un array) por correo en una caja pequeña (una cookie).

1. `serialize()` (El embalaje)

Esta función toma una estructura de datos compleja (como un array de tareas) y la convierte en una **cadena de texto plana** que conserva toda la estructura y la información.

- **¿Para qué sirve?** Para "congelar" el array y poder guardarlo en un archivo, una base de datos o, en nuestro caso, una **cookie**.
- **Ejemplo:** Si tienes este array:

  ```
  $tareas = array("Comprar pan", "Estudiar PHP");
  ```

  Al hacer `serialize($tareas)`, PHP lo convierte en algo parecido a esto: `a:2:{i:0;s:11:"Comprar pan";i:1;s:12:"Estudiar PHP";}` *(Donde `a:2` significa "array de 2 elementos", `s:11` significa "string de 11 caracteres", etc.)*. Este texto sí cabe en una cookie.

2. `unserialize()` (El desembalaje)

Esta función hace el proceso inverso. Toma esa cadena de texto "extraña" que guardamos en la cookie y la vuelve a convertir en un **array de PHP funcional**.

- **¿Para qué sirve?** Para poder volver a usar funciones como `foreach`, `array_push()` o acceder a los índices después de haber recuperado el dato de la cookie.
- **Ejemplo:**

  ```
  $texto_cookie = $_COOKIE['mis_tareas']; // Recibimos el texto plano
  $tareas = unserialize($texto_cookie);  // Lo volvemos a convertir en array
  echo $tareas[0]; // Imprimirá: Comprar pan
  ```

Resumen del ciclo en tu ejercicio:

1. **Lectura:** El navegador nos envía la cookie como texto → usamos `unserialize()` para trabajar con ella como un **array**.
2. **Modificación:** Añadimos tareas con `array_push()` o borramos con `array_splice()`.
3. **Escritura:** Antes de terminar, usamos `serialize()` para convertir el array modificado en **texto** → lo guardamos con `setcookie()`.

Nota moderna: Hoy en día se utiliza mucho también `json_encode()` y `json_decode()`, que hacen algo muy similar pero en formato JSON, que es un estándar compatible con otros lenguajes como JavaScript. Sin embargo, para ejercicios de PHP puro, `serialize` es la herramienta clásica.

Y es necesario que la use en el ejercicio?

Sí, es **totalmente necesario** si quieres almacenar más de una tarea en una sola cookie.

La razón es técnica: la superglobal `$_COOKIE` es un array asociativo donde cada "llave" (el nombre de la cookie) solo puede contener un **string** (cadena de texto) como valor.

¿Por qué no puedes prescindir de ello?

Si intentas hacer esto:

```
setcookie("mis_tareas", $array_de_tareas); // ESTO DARÁ ERROR
```

PHP generará un aviso (*Warning*) porque no sabe cómo convertir automáticamente un conjunto de datos (array) en un solo texto para enviarlo al navegador.

Comparativa: Sin vs. Con Serialización

Acción	Sin `serialize()`	Con `serialize()`
Almacenamiento	Solo podrías guardar una tarea (la última).	Guardas el array completo con todas las tareas.
Recuperación	Recibes un texto plano.	Recuperas un array listo para usar en un `foreach`.
Capacidad	No puedes usar `array_push()`.	Puedes usar todas las funciones de arrays de PHP.

¿Cuándo NO sería necesario?

Solo podrías evitar el uso de `serialize` en dos casos:

1. Si la lista tuviera **solo una tarea** (pero el ejercicio pide una "lista" o "to-do list").
2. Si crearas **una cookie distinta para cada tarea** (por ejemplo: `tarea1`, `tarea2`, `tarea3`). Esto sería muy ineficiente y difícil de gestionar.

En resumen

Para cumplir con el requisito de **"almacenar dicha tarea internamente dentro de un array"** y **"usar la función array_push"**, el flujo obligatorio es:

1. **Leer** la cookie y convertirla en array con `unserialize()`.
2. **Modificar** el array con `array_push()`.
3. **Guardar** el array convirtiéndolo en texto con `serialize()` para que `setcookie()` lo acepte.

¿Quieres que te ayude a integrar estas funciones en alguna parte específica de tu código actual?

Que hace aqui unserialize? if (isset($_COOKIE[$cookie_name])) {

$tareas = unserialize($_COOKIE[$cookie_name]);

} else {

$tareas = array();

}

En ese bloque de código, **`unserialize`** tiene la función de "reconstruir" tus datos para que PHP pueda entenderlos de nuevo como una lista.

Aquí tienes el desglose paso a paso de lo que ocurre en esas líneas:

1. El problema: El navegador solo habla "texto"

Cuando guardas la lista de tareas en una cookie, el navegador no sabe qué es un array de PHP. Para poder enviárselo al navegador, primero tuviste que convertir ese array en una cadena de texto plana (un proceso llamado **serialización**).

2. La solución: `unserialize` (El traductor)

Cuando el usuario vuelve a cargar la página, PHP recibe esa cadena de texto desde la cookie `$_COOKIE[$cookie_name]`.

- **Si la cookie existe:** `unserialize` toma ese texto (que parece un código extraño, ej: `a:2:{i:0;s:4:"leer";...}`) y lo transforma de nuevo en un **array real de PHP**.
- **El resultado:** La variable `$tareas` vuelve a ser un array sobre el que puedes usar `array_push()` o un `foreach` para mostrar la lista en el HTML.

3. El `else`: El plan de seguridad

Si es la primera vez que el usuario entra o si no hay tareas guardadas:

- `isset($_COOKIE[$cookie_name])` dará **falso**.
- El código ejecuta el `else` y crea un **array vacío** (`array()`).
- Esto es fundamental para evitar errores; si no lo hicieras, al intentar añadir una tarea con `array_push()`, PHP se quejaría de que la variable `$tareas` no existe o no es un array.

¿Te gustaría que comprobemos si tu archivo `gestion_cookies.php` está guardando correctamente la cookie para que este `unserialize` no falle?

y unserialize aquí? if (isset($_COOKIE[$cookie_name])) {

$tareas = unserialize($_COOKIE[$cookie_name]);

} else {

$tareas = array();

}

En ese bloque de código, **`unserialize()`** actúa como un "traductor" o "reconstructor" de datos.

Aquí tienes la explicación detallada de por qué es vital en ese punto:

1. El estado de la información en la Cookie

Las cookies solo pueden almacenar **texto plano (strings)**. Como tú quieres guardar una lista (un array de PHP), no puedes simplemente meter el array en la cookie porque el navegador no entendería la estructura. Para guardarlo previamente, PHP lo convirtió en una cadena de texto larga y codificada mediante `serialize()`.

2. ¿Qué hace `unserialize()` exactamente?

Cuando el navegador le devuelve la cookie a PHP (al cargar la página), la recibe como ese texto extraño. `unserialize()` toma ese texto y lo vuelve a convertir en un **objeto o array de PHP funcional**.

3. El flujo paso a paso en tu código:

- **`if (isset($_COOKIE[$cookie_name]))`**: Primero comprueba si el navegador ya tiene guardada esa cookie.
- **`$tareas = unserialize(...)`**:
  - Si la cookie existe, PHP lee el texto.

    - `unserialize` lo "desempaqueta".
    - Ahora la variable `$tareas` es un **array real**.
    - Gracias a esto, después podrás usar `foreach ($tareas ...)` para mostrar la lista o `array_push($tareas, ...)` para añadir más cosas.
- **`else { $tareas = array(); }`**: Si la cookie no existe (es la primera vez que entras), crea un array vacío para que el resto del código no falle al intentar leer una variable inexistente.

Resumen visual:

1. **En la Cookie:** El dato es una frase ilegible como `a:2:{i:0;s:4:"Hola";i:1;s:5:"Adiós";}`.
2. **Después de `unserialize()`:** El dato vuelve a ser:
    - `[0] => "Hola"`
    - `[1] => "Adiós"`

¿Te gustaría ver cómo quedaría el código si usaras `json_decode` en lugar de `unserialize`, que es una alternativa más moderna?