

## Unknown Title

---



## Conversación con Gemini

Realiza esta tarea, intentando parecerle lo máximo posible a la versión hecha con "session". debes decir porque se hace de manera diferente. "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."



- Consulta realizada correctamente

Para adaptar tu ejercicio de **Sesiones a Cookies**, el cambio fundamental radica en cómo se almacenan y transportan los datos. Mientras que las sesiones guardan la información en el servidor, las cookies residen en el navegador del usuario.

## ¿Por qué se hace de manera diferente?

La principal diferencia es que **las cookies solo pueden almacenar texto (strings)**.

- **En Sesiones:** Podías guardar el array directamente con `$_SESSION['tareas'] = array();`
- **En Cookies:** No puedes guardar un array de PHP directamente. Debes "serializarlo" (convertirlo a una cadena de texto) usando `json_encode()` para guardarlo, y "deserializarlo" con `json_decode()` para volver a trabajar con él como un array. Además, las cookies deben enviarse en las cabeceras HTTP antes de cualquier salida HTML.

## 1. arrays.php (Vista principal)

Aquí leemos la cookie, la convertimos de texto a array y mostramos el formulario.

```
<?php
// Comprobamos si existe la cookie 'tareas'.
// Si existe, decodificamos el JSON a un array de PHP. Si no, creamos un
array vacío.
$tareas = isset($_COOKIE['tareas']) ? json_decode($_COOKIE['tareas'], true)
: 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>

    <h2>Nueva tarea</h2>
    <form action="gestion_cookies.php" method="POST">
        <p>
            <label for="tarea">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); // Seguridad básica
            echo "</p>";
        }
        echo "<input type='submit' value='Borrar Marcadas' />";
    } else {
        echo "<p>No hay tareas pendientes.</p>";
    }
    ?>
</form>

<br>
<a href="logout_cookies.php">Borrar todas las cookies</a>
</body>
</html>

```

## 2. gestion\_cookies.php (Lógica de procesamiento)

Este archivo maneja la lógica de array\_push y el borrado, guardando el resultado final de vuelta en la cookie.

```

<?php
// 1. Recuperamos las tareas actuales de la cookie
$tareas = isset($_COOKIE['tareas']) ? json_decode($_COOKIE['tareas'], true)
: array();

// AÑADIR TAREA
if (isset($_POST['tarea']) && !empty(trim($_POST['tarea']))) {
    $nueva_tarea = $_POST['tarea'];

    // Usamos array_push como pide el ejercicio
    array_push($tareas, $nueva_tarea);

```

```

// Guardamos el array actualizado en la cookie (expira en 1 día)
setcookie('tareas', json_encode($tareas), time() + 86400, "/");

header("Location: arrays.php");
exit();
}

// BORRAR TAREAS
elseif (isset($_POST['borrar']) && is_array($_POST['borrar'])) {
    $indices_a_borrar = $_POST['borrar'];

    // Igual que en sesiones: orden descendente para no alterar índices al
    // borrar
    rsort($indices_a_borrar);

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

    // Actualizamos la cookie con el array resultante
    setcookie('tareas', json_encode($tareas), time() + 86400, "/");

    header("Location: arrays.php");
    exit();
}

else {
    header("Location: arrays.php");
    exit();
}
?>

```

### **3. logout\_cookies.php (Limpiar datos)**

Para borrar una cookie, se debe configurar con una fecha de expiración en el pasado.

```

<?php
    // Para eliminar la cookie, la ponemos con tiempo negativo
    setcookie('tareas', '', time() - 3600, "/");
    header("Location: arrays.php");
    exit();
?>

```

### **Resumen de cambios técnicos**

1. **Persistencia:** En lugar de `$_SESSION`, usamos `setcookie()` para escribir y `$_COOKIE` para leer.
2. **Formato:** Usamos `json_encode()` porque `setcookie` no acepta arrays directamente.
3. **Parámetros:** `setcookie` requiere el nombre, el valor (string), el tiempo de expiración y la ruta (" / " para que esté disponible en toda la web).

¿Te gustaría que añadiera alguna validación extra o te explicara más a fondo cómo funciona la serialización JSON?

En vez de usar la serialización con Json, hazlo con algo más sencillo. No puedo poner algo tan avanzado: `$tareas = isset($_COOKIE['tareas']) ? json_decode($_COOKIE['tareas'], true) : array();`



Entiendo perfectamente. Si queremos evitar funciones más avanzadas como `json_encode` o `json_decode`, la forma más sencilla y "clásica" de hacerlo es tratando la cookie como una **cadena de texto larga** donde las tareas se separan por un símbolo especial (por ejemplo, una coma o una barra vertical |).

Para ello, usaremos dos funciones muy básicas de PHP:

1. **`explode()`:** Convierte un texto en un array (separa por el símbolo).
2. **`implode()`:** Convierte un array en texto (une con el símbolo).

## ¿Por qué se hace de manera diferente?

A diferencia de las sesiones, que guardan los datos automáticamente en el servidor, las cookies solo guardan **texto plano** en el navegador del usuario. Por eso, antes de guardar el array de tareas, debemos "pegar" todas las tareas en una sola frase larga, y al leerlas, debemos "cortar" esa frase para volver a tener la lista.

---

### 1. arrays.php (Vista principal)

```
<?php
// Si la cookie existe, cortamos el texto por el símbolo "|" para crear el
array.
// Si no existe, creamos un array vacío.
if (isset($_COOKIE['tareas']) && $_COOKIE['tareas'] != "") {
    $tareas = explode("|", $_COOKIE['tareas']);
} else {
    $tareas = array();
}
?>
```

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejercicio 3 - Cookies Sencillo</title>
</head>
<body>
    <h1>Lista de Tareas (con Cookies)</h1>

    <form action="gestion_cookies.php" method="POST">
        <p>
            <label>Nueva Tarea:</label>
            <input type="text" name="tarea" placeholder="Escribe aquí">
            <input type="submit" value="Añadir">
        </p>
    </form>

    <form action="gestion_cookies.php" method="POST">
        <h2>Tareas pendientes</h2>
        <?php
            foreach ($tareas as $indice => $tarea) {
                echo "<p>";
                // El valor sigue siendo el índice numérico
                echo "<input type='checkbox' name='borrar[]' value='$indice'>
$tarea";
                echo "</p>";
            }
        ?>
        <input type="submit" value="Borrar Marcadas">
    </form>

    <br>
    <a href="logout_cookies.php">Limpiar todo</a>
</body>
</html>

```

## 2. gestion\_cookies.php (Lógica)

```

<?php
// Recuperamos lo que ya había en la cookie y lo convertimos a array
if (isset($_COOKIE['tareas']) && $_COOKIE['tareas'] != "") {
    $tareas = explode("|", $_COOKIE['tareas']);
} else {

```

```

$tareas = array();
}

// AÑADIR
if (isset($_POST['tarea']) && $_POST['tarea'] != "") {
    // Usamos array_push como pide el ejercicio
    array_push($tareas, $_POST['tarea']);

    // Convertimos el array a texto separado por "|" para guardarla en la
    cookie
    $texto_cookie = implode("|", $tareas);
    setcookie("tareas", $texto_cookie, time() + 3600, "/");

    header("Location: arrays.php");

// BORRAR
} elseif (isset($_POST['borrar'])) {
    $indices_a_borrar = $_POST['borrar'];

    // Ordenamos de mayor a menor para no desordenar el array al borrar
    rsort($indices_a_borrar);

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

    // Guardamos el array actualizado (si queda vacío, borramos la cookie)
    $texto_cookie = implode("|", $tareas);
    setcookie("tareas", $texto_cookie, time() + 3600, "/");

    header("Location: arrays.php");
} else {
    header("Location: arrays.php");
}
?>

```

### 3. logout\_cookies.php (Cerrar sesión)

```

<?php
    // Para borrar, ponemos la fecha en el pasado (hace una hora)
    setcookie("tareas", "", time() - 3600, "/");
    header("Location: arrays.php");
?>

```

## Resumen de la lógica "sencilla":

- **Para leer:** `explode(" | ", $_COOKIE['tareas'])` → De "Pan|Leche" pasa a [ 'Pan' , 'Leche' ].
- **Para guardar:** `implode(" | ", $tareas)` → De [ 'Pan' , 'Leche' ] pasa a "Pan|Leche".

¿Te parece que este método con `explode` e `implode` encaja mejor con lo que has dado en clase?

Si, me parece mejor. Ahora debes explicar muy bien porque en `session` se hace de una manera y aquí hay que usar este metodo



La principal diferencia entre usar **sesiones** y **cookies** radica en **dónde reside la información y qué tipo de datos** permite guardar cada una.

A continuación te explico detalladamente por qué el código cambia de una versión a otra:

### 1. El lugar de almacenamiento

- **En Sesiones:** Los datos se guardan en el **servidor**. El navegador solo guarda un ID (identificador) y el servidor se encarga de gestionar el array complejo de tareas por nosotros.
- **En Cookies:** Los datos se guardan directamente en el **navegador del usuario**. Como el servidor no tiene el control total, nosotros debemos enviar y recibir el "paquete" de datos en cada visita.

### 2. Limitación de tipos de datos (El motivo del `implode` y `explode`)

Esta es la razón técnica más importante:

- **Sesiones (Flexibles):** PHP permite guardar un **array directamente** en la sesión. Puedes hacer `array_push` sobre la variable de sesión y PHP se encarga de mantener la estructura de lista internamente.
- **Cookies (Solo texto):** Las cookies son como pequeñas notas de texto; **no entienden qué es un array**. Solo pueden almacenar cadenas de caracteres (strings).
  - Si intentas guardar un array directamente en una cookie, PHP te dará un error o guardará la palabra "Array", perdiendo toda la información.
  - **La solución:** Usamos `implode(" | ", $tareas)` para "aplanar" la lista y convertirla en una sola frase larga (ej: "Comprar pan|Estudiar PHP") para que quepa en la cookie.
  - **Al leer:** Usamos `explode(" | ", $_COOKIE['tareas'])` para "cortar" esa frase por los palitos y reconstruir nuestro array de tareas en PHP.

### 3. El orden de las cabeceras (Headers)

- **Sesiones:** Puedes modificar `$_SESSION` en cualquier parte del código PHP una vez iniciado el archivo.
- **Cookies:** Al usar `setcookie()`, PHP está enviando una instrucción en la cabecera de la respuesta HTTP. Esto significa que, idealmente, las cookies deben gestionarse antes de que se escriba cualquier texto o etiqueta HTML en la pantalla (como ocurre en tu archivo `gestion_cookies.php`, que procesa y luego redirige).

## 4. Persistencia y el "Logout"

- **Sesiones:** Para borrar la sesión basta con usar `unset()` o `session_destroy()` para eliminar los datos del servidor.
- **Cookies:** Como la cookie vive en el ordenador del usuario, la única forma que tiene el servidor de "borrarla" es enviando una orden al navegador para que la caduque, poniendo una fecha de expiración en el pasado (`time() - 3600`).

**¿Te gustaría que te ayude a añadir algún comentario extra en el código para que lo entiendas mejor cuando lo repases?**

```
que hace aqui el explode? if (isset($_COOKIE['tareas']) && $_COOKIE['tareas'] != "") {
    $tareas = explode("|", $_COOKIE['tareas']);
} else {
    $tareas = array();
}
?>
```



En este fragmento de código, el `explode` cumple la función de **traductor**. Su trabajo es transformar el texto plano guardado en la cookie en un array de PHP para que puedas manipular las tareas una por una.

Aquí tienes el desglose de lo que sucede paso a paso:

### 1. El problema: El formato de la Cookie

Como las cookies solo pueden guardar texto, para guardar varias tareas las hemos "pegado" usando un separador (en este caso el símbolo | ).

- En el navegador, tu cookie se ve así: "Comprar pan|Estudiar PHP|Ir al gimnasio".

### 2. La solución: ¿Qué hace `explode(" | ", ...)`?

La función explode busca ese símbolo separador (|) y corta la cadena de texto cada vez que lo encuentra.

- **Entrada (String):** "Comprar pan|Estudiar PHP|Ir al gimnasio".
- **Acción:** Detecta el "palito" | y separa las palabras.
- **Salida (Array):** 1. [0] => "Comprar pan" 2. [1] => "Estudiar PHP" 3. [2] => "Ir al gimnasio"

### 3. El flujo del if

El código que has puesto funciona como un seguro:

- **isset(\$\_COOKIE['tareas'])**: Comprueba si la cookie existe en el ordenador del usuario.
- **&& \$\_COOKIE['tareas'] != ""**: Comprueba que la cookie no esté vacía.
- **Si se cumple:** Ejecuta el explode para que la variable \$tareas se convierta en un array con el que PHP puede trabajar (por ejemplo, para hacer un foreach y mostrar la lista).
- **Si no se cumple (el else):** Crea un array() vacío para que el resto del programa no dé error al intentar leer una lista que aún no existe.

---

### Comparación con Sesiones

En tu ejercicio original de **sesiones**, no necesitabas este paso porque el array ya vivía "cocinado" y listo en el servidor. Con **cookies**, cada vez que la página carga, tienes que volver a "cocinar" el array partiendo del texto que te envía el navegador.

¿Te gustaría que viéramos un ejemplo de cómo se vería la cookie en tu navegador mientras usas la aplicación?

```
y que hace aqui implode? if (isset($_POST['tarea']) && $_POST['tarea'] != "") {  
    // Usamos array_push para poner el valor en la última posición del array.  
    array_push($tareas, $_POST['tarea']);  
    // Convertimos el array a texto separado por "|" para guardarlo en la cookie  
    $texto_cookie = implode("|", $tareas);  
    setcookie("tareas", $texto_cookie, time() + 3600, "/");  
    header("Location: arrays.php");  
  
    // BORRAR
```



Si explode funcionaba como un "cuchillo" para trocear el texto, **implode** funciona como el "pegamento".

Su función es coger todas las piezas de tu array y unirlas en una sola frase larga para que puedan ser guardadas legalmente dentro de una cookie.

## ¿Qué sucede paso a paso en ese código?

1. **array\_push(\$tareas, \$\_POST['tarea'])**: Primero, añades la nueva tarea que viene del formulario al final de tu array \$tareas. En este punto, tienes un objeto tipo array que PHP entiende, pero el navegador no.
2. **implode(" | ", \$tareas)**: Aquí es donde ocurre la transformación. Esta función toma cada elemento del array y los pega uno tras otro, poniendo el símbolo | entre ellos.
  - **Si tu array es:** ["Lavar coche", "Comprar pan"]
  - **El resultado de implode es:** "Lavar coche|Comprar pan"
3. **setcookie(...)**: Ahora que tienes una cadena de **texto plano** (el \$texto\_cookie), ya puedes enviarla al navegador para que se guarde en su memoria.

---

## La diferencia clave con las Sesiones

- **En Sesiones:** No necesitabas "pegar" nada. Podías guardar el array entero directamente en \$\_SESSION['tareas'] porque el servidor sabe manejar estructuras complejas.
- **En Cookies:** Estás obligado a usar **implode** antes de guardar los datos. Es como si para guardar una baraja de cartas en un sobre muy fino (la cookie), tuvieras que pegarlas todas una detrás de otra en una tira larga de papel. Cuando quieras volver a jugar (en arrays.php), tendrás que usar **explode** para separar las cartas de nuevo.

## Resumen visual:

- **De Array a Texto (Guardar):** implode (une con |).
- **De Texto a Array (Leer):** explode (separa por |).

¿Te gustaría que te mostrara cómo cambiar el separador | por otro símbolo o cómo afectaría esto si el usuario escribe ese mismo símbolo en su tarea?

que hace aqui array\_splice: elseif (isset(\$\_POST['borrar'])) {

```
$indices_a_borrar = $_POST['borrar'];
```

```

// Ordenamos de mayor a menor para no desordenar el array al borrar
rsort($indices_a_borrar);

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

// Guardamos el array actualizado (si queda vacío, borramos la cookie)
$texto_cookie = implode("|", $tareas);

setcookie("tareas", $texto_cookie, time() + 3600, "/");

header("Location: arrays.php");

```



La función `array_splice()` es la encargada de **extirpar** o eliminar un elemento específico del array de tareas basándose en su posición (índice).

A diferencia de otras funciones que solo borran el valor, `array_splice()` reordena el array automáticamente para que no queden "huecos" vacíos en la lista.

## ¿Cómo funciona exactamente en tu código?

La sintaxis que usas es: `array_splice($array, $posicion, $cantidad)`.

- **\$tareas**: Es el array donde están todas tus tareas (ya convertido desde la cookie).
- **\$indice**: Es el número de la posición que el usuario marcó en el checkbox (por ejemplo, la tarea 0, la 1, etc.).
- **1**: Indica que solo queremos borrar **un** elemento a partir de esa posición.

## ¿Por qué es tan importante el `rsort()` antes del `splice`?

Este es el punto más crítico de la lógica tanto en sesiones como en cookies:

- El problema:** Si tienes 3 tareas (0, 1 y 2) y borras la **número 0**, la tarea que antes era la "1" ahora pasa a ser la "0" automáticamente. Si intentas borrar la "1" justo después, estarás borrando la tarea equivocada porque los índices se han movido.
- La solución:** Al usar `rsort($indices_a_borrar)`, ordenas los números de mayor a menor (ejemplo: borrar primero la 5, luego la 2, luego la 0).
- El resultado:** Al empezar borrando por el final, las posiciones de las tareas que están al principio **no cambian**, y el borrado es preciso y sin errores.

## Diferencia con Sesiones

En este paso, la lógica es **idéntica** a la de las sesiones. La única diferencia real ocurre justo después: en sesiones el cambio ya queda guardado en el servidor, pero en cookies, después del `array_splice`, tienes que volver a usar `implode` para guardar el resultado final en el navegador del usuario.

¿Te gustaría que hiciéramos una prueba imaginaria de qué pasaría si borráramos la tarea 0 sin usar `rsort`?

C

Gemini puede cometer errores, incluso sobre personas, así que verifica sus respuestas. [Tu privacidad y GeminiSe abre en una ventana nueva](#)