**Ejercicios de Auditoría de Sistemas con Python**

# 1. Generar un Resumen de Logs del Sistema

## Objetivo:

Resumir eventos registrados en un archivo de logs (INFO, WARNING, ERROR) para identificar posibles problemas en el sistema.

El objetivo principal de generar un resumen de registros del sistema es **proporcionar una visión concisa y comprensible del comportamiento y estado de un sistema a lo largo del tiempo**. Esto permite a los administradores, desarrolladores o equipos de operaciones:

* **Identificar rápidamente problemas o anomalías:** Detectar errores, fallos, cuellos de botella de rendimiento o intentos de acceso no autorizados.
* **Monitorear el rendimiento:** Observar tendencias en el uso de recursos, tiempos de respuesta y carga del sistema.
* **Diagnosticar la causa raíz:** Facilitar la investigación de incidentes al correlacionar eventos y rastrear secuencias de acciones.
* **Cumplir con requisitos de auditoría y seguridad:** Mantener un registro de actividades para multas de cumplimiento y análisis forense.
* **Optimizar el sistema:** Tomar decisiones informadas sobre configuraciones, actualizaciones o mejoras basadas en patrones de uso.

En esencia, el resumen transforma una gran cantidad de información cruda y detallada en datos procesables y útiles para la toma de decisiones.

## Datos de Prueba:

Para generar un resumen de registros, los datos de prueba deben provenir de **archivos de registro reales generados por sistemas operativos, aplicaciones, servidores web, bases de datos o dispositivos de red**. La clave es que estos registros contienen eventos con marca de tiempo que reflejan diversas actividades.

Aquí te detallamos algunas fuentes comunes de donde podrías obtener estos datos de prueba:

* **Sistemas Operativos (Linux/Unix, Windows):**
  + **Linux/Unix:** /var/log/syslog , /var/log/auth.log, /var/log/kern.log, registros de Apache (access.log, error.log), registros de Nginx, registros de bases de datos (PostgreSQL, MySQL).
  + **Windows:** Registros de eventos de Windows (Aplicación, Seguridad, Sistema, Configuración, Eventos reenviados), que se pueden acceder a través del Visor de Eventos o exportar.
* **Servidores web:**
  + **Apache:** access.log (registra cada solicitud HTTP) y error.log(registra errores del servidor).
  + **Nginx:** access.log y error.log.
* **Servidores de Aplicaciones:** Logs generados por Tomcat, JBoss, WebLogic, o cualquier aplicación personalizada.
* **Bases de datos:** Logs de MySQL, PostgreSQL, SQL Server (logs de errores, logs de consultas lentas, logs de transacciones).
* **Dispositivos de Red:** Routers, firewalls, switchs pueden generar registros a través de syslog.
* **Contenedores y Orquestadores:** Logs de Docker, Kubernetes.

Para pruebas, lo ideal es usar **muestras representativas** de estos archivos, que incluyan tanto eventos normales como eventos que indicarían un problema (errores, advertencias, intentos de inicio de sesión fallidos). Si no tienes acceso a sistemas reales, puedes buscar conjuntos de datos de registros públicos o generar registros sintéticos que simulen el comportamiento de un sistema.

## Código:

def resumen\_logs(ruta\_archivo):  
 with open(ruta\_archivo, 'r') as f:  
 lineas = f.readlines()  
   
 errores = [line for line in lineas if "ERROR" in line]  
 advertencias = [line for line in lineas if "WARNING" in line]  
 info = [line for line in lineas if "INFO" in line]  
   
 return {  
 "Total líneas": len(lineas),  
 "Errores": len(errores),  
 "Advertencias": len(advertencias),  
 "Info": len(info)  
 }  
  
print(resumen\_logs("log\_sistema.txt"))

## Ejemplos de Archivos Log

A continuación, te muestro ejemplos de fragmentos de diferentes tipos de archivos de registro para que tengas una idea de su estructura y contenido:

### 1. Registro del sistema Linux (syslog o auth.log)

Jul 8 14:30:01 mymachine CRON[12345]: (root) CMD (command -v -- systemd >/dev/null && systemctl restart ssh)

Jul 8 14:30:05 mymachine sshd[12346]: Accepted password for user1 from 192.168.1.100 port 54321 ssh2

Jul 8 14:30:10 mymachine kernel: [12345.6789] Out of memory: Kill process 5432 (apache2) score 987 or sacrifice child

Jul 8 14:30:15 mymachine sudo[12347]: user2 : TTY=pts/0 ; PWD=/home/user2 ; USER=root ; COMMAND=/usr/bin/apt update

Jul 8 14:30:20 mymachine sshd[12348]: Failed password for invalid\_user from 203.0.113.5 port 22 ssh2

Jul 8 14:30:25 mymachine systemd[1]: Started Session 123 of user user1.

**Elementos clave:** Fecha, Hora, Nombre de host, Proceso/Aplicación, Mensaje del evento.

### 2. Registro del servidor web Apache (access.log)

192.168.1.1 - - [08/Jul/2025:15:00:01 -0500] "GET /index.html HTTP/1.1" 200 1234 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.0.0 Safari/537.36"

192.168.1.2 - - [08/Jul/2025:15:00:05 -0500] "POST /api/data HTTP/1.1" 200 56 "http://example.com/app" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.0 Safari/605.1.15"

192.168.1.3 - - [08/Jul/2025:15:00:10 -0500] "GET /nonexistent\_page HTTP/1.1" 404 200 "-" "curl/7.81.0"

192.168.1.4 - - [08/Jul/2025:15:00:15 -0500] "GET /admin/setup HTTP/1.1" 401 500 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)"

**Elementos clave:** IP del cliente, Fecha y Hora, Método HTTP, Ruta solicitada, Versión HTTP, Código de estado HTTP, Tamaño de la respuesta, Referer, User-Agent.

### 3. Registro de errores de Apache (error.log)

[Tue Jul 08 15:05:01.123456 2025] [core:error] [pid 12345:tid 67890] [client 192.168.1.5:12345] AH00037: File does not exist: /var/www/html/favicon.ico

[Tue Jul 08 15:05:10.789012 2025] [php:error] [pid 12345:tid 67890] [client 192.168.1.6:54321] PHP Fatal error: Uncaught Error: Call to undefined function nonExistentFunction() in /var/www/html/app.php:10

Stack trace:

#0 {main}

thrown in /var/www/html/app.php on line 10

**Elementos clave:** Fecha y Hora, Nivel de log (error, warn, info), Módulo, PID/TID, Cliente, Mensaje de error, (opcionalmente) Stack trace.

### 4. Registro de Base de Datos (ej. PostgreSQL - pg\_log)

2025-07-08 15:10:01.123 EDT [12345]: [1-1] user=dbuser,database=mydb,client=192.168.1.7 LOG: duration: 0.543 ms execute <unnamed>: SELECT \* FROM users WHERE id = $1

2025-07-08 15:10:05.678 EDT [12346]: [2-1] user=appuser,database=appdb,client=192.168.1.8 ERROR: duplicate key value violates unique constraint "users\_pkey"

2025-07-08 15:10:10.901 EDT [12347]: [3-1] user=admin,database=admin\_db,client=192.168.1.9 LOG: checkpoint complete

**Elementos clave:** Fecha y Hora, Zona horaria, PID, Nivel de registro (LOG, ERROR, FATAL), Usuario de BD, Base de datos, Cliente, Mensaje del evento (ej. duración de consulta, error de duplicado).

# 2. Validar la Integridad de Archivos usando Hashes

## Objetivo:

El **objetivo** de este ejercicio es **verificar que un archivo no haya sido modificado o corrompido desde que se generó su hash original**. Esto se logra comparando el hash calculado de un archivo en un momento dado con un hash de referencia previamente establecido.

En términos prácticos, busca confirmar:

* **Autenticidad:** Asegurar que el archivo es el original y no una versión alterada o maliciosa.
* **Consistencia:** Garantizar que no hubo errores de transmisión o almacenamiento que dañen el archivo.

## Datos de Prueba: ¿De dónde deben provenir?

Para este ejercicio, los **datos de prueba** deben consistir en un **conjunto de archivos junto con sus hashes correspondientes**, generados previamente. Es crucial tener tanto archivos "buenos" (cuya integridad se espera que sea válida) como archivos "alterados" o "corruptos" (que deben fallar la validación).

Puedes obtener estos datos de varias maneras:

1. **Manual de creación:**
   * **Archivos Originales:** Crea varios archivos de texto simple, documentos (PDF, DOCX), imágenes (JPG, PNG), etc. Asegúrate de que sean de diferentes tamaños.
   * **Generación de Hashes de Referencia:** Utilice una herramienta de línea de comandos (como certutilen Windows o shasum/ md5sumen Linux/macOS) para generar los hashes (MD5, SHA-256) de estos archivos originales.
   * **Archivos alterados:** Para cada archivo original, cree una copia y modifique ligeramente la copia (por ejemplo, agregue un espacio en blanco, cambie una letra, etc.). El hash de estas copias alteradas **debe ser diferente** al original.
2. **Descargas con Hashes Publicados:** Muchos sitios web de software o distribuciones de Linux publican los hashes (checksums) de sus archivos descargables. Puedes descargar un archivo y usar el hash publicado como tu hash de referencia. Luego, podrías modificar el archivo descargado para simular una corrupción.

**Ejemplo de cómo generar un hash (SHA-256) en la terminal:**

* **Linux/macOS:**

Intento

shasum -a 256 nombre\_del\_archivo.txt

* **Ventanas (PowerShell):**

PowerShell

Get-FileHash -Path "C:\ruta\al\nombre\_del\_archivo.txt" -Algorithm SHA256

### Ejemplo de archivos y hashes

Imagina que tienes los siguientes archivos y sus hashes de referencia:

### Archivo Original (Integridad Válida Esperada):

* **Nombre del archivo:** documento\_secreto.txt
* **Contenido (Ejemplo):**
* Este es un documento importante.
* Contiene información confidencial.
* **Hash SHA-256 de Referencia:** d62a3e0f9c8b1a2d3f4e5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d

### Archivo Alterado (Integridad Inválida Esperada):

* **Nombre del archivo:** documento\_secreto\_modificado.txt
* **Contenido (Ejemplo - una pequeña modificación):**
* Este es un documento importante.
* Contiene información confidencial.
* (Modificado)
* **Hash SHA-256 de Referencia (del original, para comparar):** d62a3e0f9c8b1a2d3f4e5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d *(Nota: El hash real calculado de este archivo modificado* ***será diferente*** *)*

### Archivo Binario (Ejemplo):

* **Nombre del archivo:** imagen\_vacaciones.jpg
* **Hash MD5 de Referencia:** e8b2a3c4d5e6f7a8b9c0d1e2f3a4b5c6

## Información para ingresar

Para cada validación, tu sistema o script deberá pedir o recibir la siguiente información:

1. **Ruta del Archivo a Validar:** La ubicación completa del archivo en tu sistema de archivos (ej., C:\MisDocumentos\documento\_secreto.txto /home/usuario/documento\_secreto.txt).
2. **Hash de Referencia Esperado:** El hash que se sabe que pertenece al archivo original y sin alteraciones.
3. **Algoritmo de Hash a Usar:** El tipo de algoritmo de hash que se debe emplear para el cálculo (ej., MD5, SHA-1, SHA-256, SHA-512).

Una vez que se ingresa esta información, tu programa deberá:

1. Calcular el hash del archivo proporcionado usando el algoritmo especificado.
2. Comparar el hash calculado con el hash de referencia.
3. Reportar el resultado: "Integridad Verificada" o "Integridad Comprometida".

## Datos de Prueba:

Archivo: archivo\_a\_verificar.txt  
Hash esperado: 5d41402abc4b2a76b9719d911017c592 (MD5 de 'hello')

## Código:

import hashlib  
  
def hash\_archivo(ruta, algoritmo='md5'):  
 h = hashlib.new(algoritmo)  
 with open(ruta, 'rb') as f:  
 while chunk := f.read(8192):  
 h.update(chunk)  
 return h.hexdigest()  
  
hash\_original = "5d41402abc4b2a76b9719d911017c592"  
hash\_actual = hash\_archivo("archivo\_a\_verificar.txt")  
  
print("Integridad OK" if hash\_original == hash\_actual else "Archivo modificado")

# 3. Analizar Permisos de Archivos (Linux)

## Objetivo:

Auditar los permisos de archivos para detectar configuraciones inseguras.

## Datos de Prueba:

Directorio: ./test\_dir  
Archivos con permisos: archivo1.txt (644), archivo2.sh (777)

## Código:

import os  
  
def analizar\_permisos(directorio):  
 for archivo in os.listdir(directorio):  
 ruta = os.path.join(directorio, archivo)  
 permisos = oct(os.stat(ruta).st\_mode)[-3:]  
 print(f"{archivo}: permisos {permisos}")  
  
analizar\_permisos("./test\_dir")

# 4. Auditar la Configuración de Red (Escaneo básico)

## Objetivo:

Detectar puertos abiertos en un host de la red local.

## Datos de Prueba:

IP: 127.0.0.1  
Puertos a escanear: 22, 80, 3306

## Código:

import socket  
  
def escanear\_puertos(ip, puertos=[22, 80, 3306]):  
 for puerto in puertos:  
 sock = socket.socket()  
 sock.settimeout(1)  
 try:  
 sock.connect((ip, puerto))  
 print(f"Puerto {puerto} abierto")  
 except:  
 print(f"Puerto {puerto} cerrado")  
 sock.close()  
  
escanear\_puertos("127.0.0.1")

# 5. Analizar Transacciones en CSV

## Objetivo:

Sumar montos financieros para verificar registros contables.

## Datos de Prueba:

transacciones.csv:  
fecha,monto  
2025-07-01,100.50  
2025-07-02,200.00  
2025-07-03,-50.00

## Código:

import csv  
  
def analizar\_csv(ruta):  
 total = 0  
 with open(ruta) as f:  
 lector = csv.DictReader(f)  
 for fila in lector:  
 total += float(fila['monto'])  
 print(f"Total de transacciones: {total:.2f}")  
  
analizar\_csv("transacciones.csv")

# 6. Verificar Disponibilidad de Servicios Web

## Objetivo:

Detectar si sitios web o servicios críticos están activos.

## Datos de Prueba:

URL: https://www.google.com

## Código:

import requests  
  
def comprobar\_servicio(url):  
 try:  
 r = requests.get(url, timeout=3)  
 print(f"{url}: Disponible ({r.status\_code})")  
 except:  
 print(f"{url}: No disponible")  
  
comprobar\_servicio("https://www.google.com")

# 7. Verificar Contraseñas según Política

## Objetivo:

Comprobar si las contraseñas cumplen con políticas de seguridad.

## Datos de Prueba:

Contraseñas: Admin123 (válida), admin (inválida)

## Código:

import re  
  
def cumple\_politica(passw):  
 if (len(passw) >= 8 and  
 re.search(r"[A-Z]", passw) and  
 re.search(r"[a-z]", passw) and  
 re.search(r"[0-9]", passw)):  
 return True  
 return False  
  
print(cumple\_politica("Admin123")) # True  
print(cumple\_politica("admin")) # False

# 8. Detectar Accesos No Autorizados

## Objetivo:

Identificar registros de accesos fallidos o sospechosos.

## Datos de Prueba:

logs\_seguridad.txt:  
LOGIN SUCCESS - usuario: admin  
FAILED LOGIN - usuario: root  
UNAUTHORIZED access attempt from IP 192.168.0.12

## Código:

def detectar\_accesos\_no\_autorizados(ruta\_logs):  
 with open(ruta\_logs) as f:  
 for linea in f:  
 if "FAILED LOGIN" in linea or "UNAUTHORIZED" in linea:  
 print("Posible acceso no autorizado:", linea.strip())  
  
detectar\_accesos\_no\_autorizados("logs\_seguridad.txt")

# 9. Verificar Cumplimiento de Políticas Internas

## Objetivo:

Revisar cumplimiento de configuraciones básicas de seguridad.

## Datos de Prueba:

config = {"firewall\_activado": False, "usuarios\_con\_root": 3}

## Código:

def verificar\_politicas(configs):  
 errores = []  
 if not configs.get("firewall\_activado"):  
 errores.append("Firewall desactivado")  
 if configs.get("usuarios\_con\_root") > 0:  
 errores.append("Usuarios con acceso root no restringido")  
 return errores or ["Todo cumple"]  
  
print("Resultado:", verificar\_politicas({  
 "firewall\_activado": False,  
 "usuarios\_con\_root": 3  
}))

# 10. Revisar Políticas de Configuración del Servidor

## Objetivo:

Automatizar auditorías en archivos de configuración del sistema.

## Datos de Prueba:

sshd\_config contiene: PermitRootLogin yes

## Código:

def revisar\_configuracion\_servidor(ruta\_conf):  
 with open(ruta\_conf) as f:  
 contenido = f.read()  
 if "PermitRootLogin yes" in contenido:  
 print("¡Peligro! Root login está permitido.")  
 else:  
 print("Configuración segura.")  
  
revisar\_configuracion\_servidor("sshd\_config")

# Ejercicios de programación en Python, ordenados desde los más sencillos hasta algunos más complejos:

1. **Imprimir "Hola, Mundo"**

# Este programa imprime la frase "Hola Mundo" en la consola.

print("Hola Mundo")

Hola Mundo

1. **Sumar dos números ingresados por el usuario**

Este código te permite ingresar números enteros o decimales y calculará su suma, además de incluir un manejo básico de errores por si se ingresa texto en lugar de números.

**Ejemplo:**

# Este programa solicita al usuario que ingrese dos números y luego calcula y muestra su suma.

try:

    # Solicitar al usuario el primer número

    # La función input() lee la entrada como una cadena, por lo que necesitamos convertirla a un tipo numérico.

    num1\_str = input("Por favor, ingresa el primer número: ")

    num1 = float(num1\_str) # Convertir la cadena a un número flotante para permitir decimales

    # Solicitar al usuario el segundo número

    num2\_str = input("Por favor, ingresa el segundo número: ")

    num2 = float(num2\_str) # Convertir la cadena a un número flotante

    # Calcular la suma de los dos números

    suma = num1 + num2

    # Imprimir el resultado de la suma

    # Usamos una f-string para formatear la salida de manera clara.

    print(f"La suma de {num1} y {num2} es: {suma}")

except ValueError:

    # Manejar el error si el usuario ingresa algo que no es un número

    print("Entrada inválida. Por favor, asegúrate de ingresar solo números.")

except Exception as e:

    # Manejar cualquier otro error inesperado

    print(f"Ocurrió un error inesperado: {e}")

Por favor, ingresa el primer número: 70

Por favor, ingresa el segundo número: 20

La suma de 70.0 y 20.0 es: 90.0

1. **Calcular la suma de todos los números en una lista**

Este código define una función sumar\_lista que toma una lista de números y devuelve su suma. Incluye dos opciones para realizar la suma: una usando un bucle for (para mayor claridad en la lógica) y otra usando la función sum() incorporada de Python (que es más concisa y eficiente). También se proporcionan ejemplos de cómo usar la función con diferentes listas.

**Ejemplo:**

# Este programa demuestra cómo calcular la suma de todos los números en una lista.

def sumar\_lista(numeros):

    """

    Calcula la suma de todos los números en una lista dada.

    Args:

        numeros (list): Una lista de números (enteros o flotantes).

    Returns:

        float/int: La suma total de los números en la lista.

    """

    # Opción 1: Usando un bucle (más explícito para entender la lógica)

    suma\_total = 0

    for numero in numeros:

        suma\_total += numero

    return suma\_total

    # Opción 2: Usando la función sum() incorporada de Python (más concisa y eficiente)

    # return sum(numeros)

# Ejemplo de uso:

mi\_lista\_de\_numeros = [10, 20, 30, 40, 50, 5.5, 12.3]

# Llamar a la función para sumar los números de la lista

resultado\_suma = sumar\_lista(mi\_lista\_de\_numeros)

# Imprimir la lista y el resultado de la suma

print(f"La lista de números es: {mi\_lista\_de\_numeros}")

print(f"La suma de todos los números en la lista es: {resultado\_suma}")

# Otro ejemplo con una lista diferente

otra\_lista = [-1, 0, 100, 25.7]

print(f"\nLa lista de números es: {otra\_lista}")

print(f"La suma de todos los números en la lista es: {sumar\_lista(otra\_lista)}")

La lista de números es: [10, 20, 30, 40, 50, 5.5, 12.3]

La suma de todos los números en la lista es: 167.8

La lista de números es: [-1, 0, 100, 25.7]

La suma de todos los números en la lista es: 124.7

1. **Crear una función que determine si un número es primo**

Esta función es\_primo toma un número entero y devuelve True si es primo y False en caso contrario. Incluye optimizaciones para verificar solo divisores impares y hasta la raíz cuadrada del número para mejorar la eficiencia. También se incluyen varios ejemplos para que puedas probarla.

**Ejemplo:**

# Este programa define una función para determinar si un número dado es primo.

def es\_primo(numero):

    """

    Determina si un número entero es primo.

    Un número primo es un número natural mayor que 1 que no tiene divisores positivos

    más que 1 y él mismo.

    Args:

        numero (int): El número entero a verificar.

    Returns:

        bool: True si el número es primo, False en caso contrario.

    """

    # Los números menores o iguales a 1 no son primos.

    if numero <= 1:

        return False

    # El 2 es el único número primo par.

    if numero == 2:

        return True

    # Si el número es par y mayor que 2, no es primo.

    if numero % 2 == 0:

        return False

    # Verificar divisores impares desde 3 hasta la raíz cuadrada del número.

    # No necesitamos verificar más allá de la raíz cuadrada porque si un número 'n'

    # tiene un divisor 'd' mayor que su raíz cuadrada, entonces también debe tener

    # un divisor 'n/d' que es menor que su raíz cuadrada.

    i = 3

    while i \* i <= numero:

        if numero % i == 0:

            return False

        i += 2  # Solo necesitamos verificar divisores impares

    # Si no se encontraron divisores, el número es primo.

    return True

# Ejemplos de uso:

print(f"¿Es 7 primo? {es\_primo(7)}")         # Debería ser True

print(f"¿Es 10 primo? {es\_primo(10)}")       # Debería ser False

print(f"¿Es 2 primo? {es\_primo(2)}")         # Debería ser True

print(f"¿Es 1 primo? {es\_primo(1)}")         # Debería ser False

print(f"¿Es 0 primo? {es\_primo(0)}")         # Debería ser False

print(f"¿Es 29 primo? {es\_primo(29)}")       # Debería ser True

print(f"¿Es 97 primo? {es\_primo(97)}")       # Debería ser True

print(f"¿Es 99 primo? {es\_primo(99)}")       # Debería ser False

¿Es 7 primo? True

¿Es 10 primo? False

¿Es 2 primo? True

¿Es 1 primo? False

¿Es 0 primo? False

¿Es 29 primo? True

¿Es 97 primo? True

¿Es 99 primo? False

1. **Desarrollar un programa que gestione una lista de tareas (to-do list)**

Este programa te permite:

* **Añadir tarea:** Ingresar una descripción para una nueva tarea.
* **Ver tareas:** Mostrar todas las tareas, indicando si están completadas o pendientes.
* **Marcar tarea como completada:** Seleccionar una tarea por su número para cambiar su estado.
* **Eliminar tarea:** Seleccionar una tarea por su número para quitarla de la lista.

El programa se ejecuta en la consola y las tareas se guardan mientras el programa está en ejecución. Si cierras el programa, la lista de tareas se reiniciará.

**Ejemplo:**

# Este programa gestiona una lista de tareas (to-do list) en memoria.

# Las tareas se almacenan en una lista de diccionarios.

# Lista global para almacenar las tareas. Cada tarea es un diccionario.

# Ejemplo: {"descripcion": "Comprar leche", "completada": False}

lista\_de\_tareas = []

def mostrar\_menu():

    """Muestra las opciones del menú principal al usuario."""

    print("\n--- GESTOR DE TAREAS ---")

    print("1. Añadir tarea")

    print("2. Ver tareas")

    print("3. Marcar tarea como completada")

    print("4. Eliminar tarea")

    print("5. Salir")

    print("------------------------")

def añadir\_tarea():

    """Solicita al usuario una descripción y añade una nueva tarea a la lista."""

    descripcion = input("Introduce la descripción de la nueva tarea: ").strip()

    if descripcion:

        tarea = {"descripcion": descripcion, "completada": False}

        lista\_de\_tareas.append(tarea)

        print(f"Tarea '{descripcion}' añadida.")

    else:

        print("La descripción de la tarea no puede estar vacía.")

def ver\_tareas():

    """Muestra todas las tareas en la lista con su estado (completada/pendiente)."""

    if not lista\_de\_tareas:

        print("No hay tareas en la lista.")

        return

    print("\n--- TUS TAREAS ---")

    for i, tarea in enumerate(lista\_de\_tareas):

        estado = "✓" if tarea["completada"] else " "

        print(f"{i + 1}. [{estado}] {tarea['descripcion']}")

    print("------------------")

def marcar\_tarea\_completada():

    """

    Permite al usuario marcar una tarea existente como completada.

    Solicita el número de la tarea de la lista.

    """

    ver\_tareas() # Muestra las tareas para que el usuario pueda elegir.

    if not lista\_de\_tareas:

        return # No hay tareas para marcar.

    try:

        num\_tarea\_str = input("Introduce el número de la tarea a marcar como completada: ")

        num\_tarea = int(num\_tarea\_str)

        # Ajustar el índice para que coincida con la lista (base 0)

        indice\_tarea = num\_tarea - 1

        if 0 <= indice\_tarea < len(lista\_de\_tareas):

            if not lista\_de\_tareas[indice\_tarea]["completada"]:

                lista\_de\_tareas[indice\_tarea]["completada"] = True

                print(f"Tarea '{lista\_de\_tareas[indice\_tarea]['descripcion']}' marcada como completada.")

            else:

                print("Esta tarea ya estaba marcada como completada.")

        else:

            print("Número de tarea inválido. Por favor, introduce un número de la lista.")

    except ValueError:

        print("Entrada inválida. Por favor, introduce un número entero.")

    except Exception as e:

        print(f"Ocurrió un error inesperado: {e}")

def eliminar\_tarea():

    """

    Permite al usuario eliminar una tarea de la lista.

    Solicita el número de la tarea a eliminar.

    """

    ver\_tareas() # Muestra las tareas para que el usuario pueda elegir.

    if not lista\_de\_tareas:

        return # No hay tareas para eliminar.

    try:

        num\_tarea\_str = input("Introduce el número de la tarea a eliminar: ")

        num\_tarea = int(num\_tarea\_str)

        # Ajustar el índice para que coincida con la lista (base 0)

        indice\_tarea = num\_tarea - 1

        if 0 <= indice\_tarea < len(lista\_de\_tareas):

            tarea\_eliminada = lista\_de\_tareas.pop(indice\_tarea)

            print(f"Tarea '{tarea\_eliminada['descripcion']}' eliminada.")

        else:

            print("Número de tarea inválido. Por favor, introduce un número de la lista.")

    except ValueError:

        print("Entrada inválida. Por favor, introduce un número entero.")

    except Exception as e:

        print(f"Ocurrió un error inesperado: {e}")

def main():

    """Función principal que ejecuta el bucle del programa."""

    while True:

        mostrar\_menu()

        opcion = input("Elige una opción: ").strip()

        if opcion == '1':

            añadir\_tarea()

        elif opcion == '2':

            ver\_tareas()

        elif opcion == '3':

            marcar\_tarea\_completada()

        elif opcion == '4':

            eliminar\_tarea()

        elif opcion == '5':

            print("¡Hasta luego! Saliendo del gestor de tareas.")

            break

        else:

            print("Opción no válida. Por favor, elige una opción del 1 al 5.")

# Punto de entrada del programa

if \_\_name\_\_ == "\_\_main\_\_":

    main()

--- GESTOR DE TAREAS ---

1. Añadir tarea

2. Ver tareas

3. Marcar tarea como completada

4. Eliminar tarea

5. Salir

------------------------

---------------------------------------------------------------------------

KeyboardInterrupt Traceback (most recent call last)

[/tmp/ipython-input-1-1064013741.py](https://localhost:8080/) in <cell line: 0>()

**113** # Punto de entrada del programa

**114** if \_\_name\_\_ == "\_\_main\_\_":

--> 115 main()

2 frames

[/usr/local/lib/python3.11/dist-packages/ipykernel/kernelbase.py](https://localhost:8080/) in \_input\_request(self, prompt, ident, parent, password)

**1217** except KeyboardInterrupt:

**1218** # re-raise KeyboardInterrupt, to truncate traceback

-> 1219 raise KeyboardInterrupt("Interrupted by user") from None

**1220** except Exception:

**1221** self.log.warning("Invalid Message:", exc\_info=True)

KeyboardInterrupt: Interrupted by user