Planificador Inteligente de Eventos

El proyecto consiste en construir una aplicación de software completa para planificar **eventos** que consumen **recursos** de un inventario limitado. Dado que los eventos tienen una duración específica en el tiempo (un inicio y un fin), el reto principal del proyecto es garantizar que no existan conflictos ni colisiones en la asignación de recursos. Por lo tanto, manejar las nociones de operaciones con intervalos de tiempo será una habilidad central para el éxito. Te enfrentarás a uno de los problemas más comunes y cruciales en el mundo real: la gestión de la disponibilidad y la resolución de conflictos.

El objetivo principal es desarrollar un motor de planificación inteligente que garantice dos cosas en todo momento:

1. Que los recursos no se asignen a más de un evento a la vez.
2. Que se respeten un conjunto de reglas y **restricciones** personalizadas que tú mismo definirás.

Tu misión se divide en dos fases clave: una de diseño creativo y otra de implementación técnica.

Diseño

Antes de escribir una sola línea de código, tu primera tarea es la de un arquitecto de software: elegirás un dominio del mundo real y modelarás sus reglas. ¿Gestionarás un hospital, una productora de cine, un laboratorio de investigación, un centro deportivo? La elección es tuya.

Dentro del dominio que elijas, deberás definir los siguientes componentes fundamentales:

Eventos

Son las actividades principales que necesitan ser planificadas en el tiempo y que requieren recursos para poder llevarse a cabo.

- **Ejemplos:** "Cirugía de Corazón Abierto", "Rodaje de la Escena 5", "Experimento de Fusión Fría", "Partido Final del Torneo".

Recursos

Es el inventario de todos los activos finitos, compartidos y reutilizables que tus eventos necesitan para ocurrir. Puedes modelar tus recursos como simples identificadores (nombres) o, si tu dominio lo requiere, añadirles atributos específicos (ej: capacidad, marca, modelo) para crear restricciones más interesantes.

- **Ejemplos:** "Quirófano 3", "Dr. Martínez (Cardiólogo)", "Cámara RED Epic-W", "Técnico de Sonido", "Microscopio Electrónico", "Pista Central".

c. Restricciones entre Recursos

Esta es la parte más creativa y el verdadero núcleo de la lógica de tu proyecto. Debes modelar un conjunto de reglas que dicten cómo los recursos pueden (o no pueden) ser combinados. Estas restricciones le darán una personalidad única a tu aplicación.

Debes implementar al menos dos tipos de restricciones:

1. **Restricción de Co-requisito (Inclusión):** Un recurso siempre requiere de otro recurso complementario para ser utilizado en un evento.
    - *Ejemplo en una productora de cine:* Una "Cámara RED" siempre debe ser asignada junto con un "Técnico de Cámara Certificado".
    - *Ejemplo en un hospital:* Una "Cirugía Robótica" siempre requiere la asignación de la "Consola Da Vinci" y un "Cirujano Certificado en Da Vinci".
2. **Restricción de Exclusión Mutua:** Si un evento utiliza un recurso de un tipo, tiene prohibido utilizar otro recurso de otro tipo en el mismo evento.
    - *Ejemplo en un laboratorio:* Un "Experimento Químico" no puede usar el "Mechero Bunsen" al mismo tiempo que el "Contenedor de Éter" por razones de seguridad.
    - *Ejemplo en un estudio de grabación:* Una sesión en la "Sala de Grabación A" no puede usar el "Micrófono de Cinta (Ribbon)", que es extremadamente sensible, si también se ha reservado la "Batería Acústica".

Nota sobre la implementación: Se espera que la lógica de validación de tus restricciones personalizadas forme parte del código de tu aplicación, aunque de forma opcional puede ser configurable por el usuario. Asegúrate de documentar y explicar muy bien estas reglas en tu archivo `README.md`.

Implementación

Una vez diseñado tu dominio, el reto técnico es escribir el código que gestione el calendario de eventos, el estado del inventario de recursos y la validación de todas las reglas. Tu aplicación debe ser robusta y manejar adecuadamente los errores. Si un usuario introduce datos incorrectos (ej. una fecha mal formada) o una operación no se puede realizar, el programa no debe fallar, sino mostrar un mensaje de error claro y descriptivo.

Operaciones y Lógica Requeridas

Planificar un Nuevo Evento

Esta será la operación central de tu programa. El usuario propondrá un evento, especificando los recursos que necesita y el intervalo de tiempo deseado. El sistema debe verificar **automáticamente** dos condiciones críticas antes de confirmar la planificación:

1. **Conflictos de Recursos:** Que ninguno de los recursos solicitados esté ya asignado a otro evento en ese mismo horario.
2. **Violación de Restricciones:** Que la combinación de recursos solicitada no viole ninguna de las reglas de co-requisito o exclusión que definiste en tu dominio.

Búsqueda Automática de Horarios ("Buscar Hueco")

Debes implementar una función inteligente que, dado un evento y los recursos que necesita, sea capaz de analizar el calendario y sugerir el **próximo intervalo de tiempo disponible** donde se pueda realizar sin conflictos ni violaciones de restricciones.

Interfaz de Usuario

La interacción con tu aplicación puede ser a través de una **interfaz de consola (CLI)** o una **interfaz gráfica (GUI) básica**. Independientemente de la opción, debe permitir al usuario realizar las siguientes acciones:

- **Listar** todos los eventos planificados.
- **Agregar** un nuevo evento, invocando toda la lógica de validación.
- **Eliminar** un evento existente, liberando sus recursos para que queden disponibles.
- **Ver Detalles** de un evento específico (qué recursos usa, a qué hora) o de un recurso (cuál es su agenda).

Persistencia de Datos

Todo el estado de la aplicación (la definición de los recursos, la lista de eventos planificados, etc.) debe poder **guardarse y cargarse desde un único archivo** (ej. en formato JSON o un formato de texto propio). Esta funcionalidad es fundamental, ya que permite que tu aplicación pueda gestionar diferentes dominios y escenarios simplemente cambiando el archivo que se carga al inicio.

4. Requisitos Técnicos y Entregables

- **Tecnología:** Python. Se recomienda el uso de la librería `datetime` para la gestión del tiempo y `json` para la persistencia de datos.
- **Entregables:**
    1. **Código Fuente:** Todos los archivos `.py` de tu proyecto, debidamente comentados y organizados.
    2. **Documento `README.md`:** Es una parte crucial del proyecto. Debe explicar claramente:
        - El dominio que elegiste y por qué.

        - Una descripción detallada de los eventos, recursos y, muy importante, **las restricciones que implementaste**, con ejemplos.
        - Instrucciones claras sobre cómo ejecutar el programa y usar sus funcionalidades.
    3. **Archivo de Datos de Ejemplo:** Un archivo de datos (`.json`, `.txt`, etc.) que demuestre el funcionamiento de tu aplicación en el dominio elegido.
    4. **Control de Versiones:** El proyecto debe ser desarrollado usando **Git** y entregado a través de un enlace a un repositorio (GitHub). Se esperan *commits* frecuentes que muestren un progreso incremental.

5. Desafíos Opcionales para Sobresalir

Si completas todos los requisitos mínimos y quieres mejorar tu calificación, puedes implementar una o más de las siguientes funcionalidades avanzadas:

- **Recursos con Cantidad (Pools de Recursos):** En lugar de que cada recurso sea único (ej. "Cámara 1", "Cámara 2"), permite que existan recursos con una cantidad disponible (ej. "Cámara", `cantidad: 5`). La lógica de conflictos deberá comprobar si quedan unidades disponibles en lugar de un simple "ocupado/libre".
- **Planificación de Eventos Recurrentes:** Añade la opción de crear eventos que se repitan automáticamente cada día, semana o mes. El sistema deberá ser capaz de planificar todas las ocurrencias futuras, validando los conflictos y restricciones para cada una de ellas de forma individual.
- **Cualquiera otra funcionalidad interesante que se te ocurra**...