Ejercicio Práctico: Aplicación Python con Tkinter, Unit Testing y CI/CD en GitHub

Objetivo

Desarrollar una pequeña aplicación Python que permita realizar operaciones básicas, aprendiendo a:

- Separar lógica de la interfaz para testearla.
- Crear y usar entornos virtuales y requirements.
- Implementar tests unitarios.
- Generar un ejecutable .exe con PyInstaller.
- Configurar GitHub para CI/CD con workflows y protección de ramas.

Enunciado

Se desea desarrollar una **calculadora de operaciones básicas** (suma, resta, multiplicación, división) con la siguiente estructura:

Punto 1: Configuración inicial y entorno virtual

1. Crea una carpeta para tu proyecto: mi_calculadora_app.
2. Dentro de la carpeta, crea un entorno virtual:
    - Windows: python -m venv .venv
    - Linux/macOS: python3 -m venv .venv
3. Activa el entorno virtual.
4. Instala las dependencias necesarias (Tkinter ya viene con Python, instalar PyInstaller).
5. Genera un archivo requirements.txt con todas las dependencias instaladas.

Punto 2: Implementar la lógica separada de la GUI

1. Crea un archivo calc.py con funciones:

```
def sumar(a, b): ...

def restar(a, b): ...

def multiplicar(a, b): ...

def dividir(a, b): ...  # lanzar ValueError si b = 0
```

2. Cada función debe validar que los parámetros sean numéricos (int o float). Si no, lanzar ValueError.

Punto 3: Crear la GUI con Tkinter

1. Crea un archivo app.py.
2. La GUI debe permitir:
    - Ingresar dos números.
    - Seleccionar la operación (suma, resta, multiplicación, división).
    - Mostrar el resultado.
3. La GUI debe **usar las funciones de calc.py** y no tener lógica compleja propia.

Punto 4: Unit Testing

1. Crea una carpeta tests/ y un archivo test_calc.py.
2. Implementa tests unitarios usando unittest:
    - Probar cada función con valores válidos.
    - Probar que se lancen errores con entradas no numéricas o divisiones por cero.
3. Ejecuta los tests localmente:
4. python -m unittest discover -v

Punto 5: Generar un ejecutable .exe con PyInstaller

1. Desde el entorno virtual, genera un ejecutable de app.py:
2. pyinstaller --onefile --windowed app.py
3. Verifica que el .exe funciona correctamente.

Punto 6: Configuración de GitHub y CI/CD

1. Inicializa un repositorio Git y sube tu proyecto a GitHub.
2. Protege la rama principal (master):
    - No permitir push directo.
    - Requerir Pull Request revisado.
    - Requerir que los tests pasen antes del merge.
3. Crea un workflow de GitHub Actions para **ejecutar tests automáticamente** en cada PR.
4. Crea un workflow que genere el .exe automáticamente **cuando se haga merge en master**.
5. Verifica que el flujo completo funcione:
    - Crear rama → hacer cambios → abrir PR → tests pasan → merge → generar .exe.

Punto 7: Bonus (Opcional)

- Agrega validación en la GUI para que no se pueda ingresar texto que no sea número.
- Implementa logging de las operaciones realizadas en un archivo log.txt.
- Añade icono personalizado al .exe con PyInstaller (--icon=icon.ico).