4. **Generador de contraseñas:** Crea contraseñas seguras y aleatorias (con el módulo `secrets`).

¡Claro! Aquí tienes una aplicación en Python para generar **contraseñas seguras y aleatorias** usando el módulo estándar **secrets**, que es el recomendado para este tipo de tareas.

Este script te permite especificar la **longitud** de la contraseña.

? Generador de Contraseñas Seguras en Python

Python

```
import secrets
import string

def generar_contrasena_segura(longitud=12):
    """
    Genera una contraseña aleatoria y segura de una longitud especificada.

    La contraseña incluye:
    - Caracteres alfanuméricos (letras mayúsculas y minúsculas, y dígitos)
    - Símbolos (puntuación)

    Args:
        longitud (int): La longitud deseada para la contraseña. Por defecto es 12.

    Returns:
        str: La contraseña generada, o None si la longitud es muy pequeña.
    """
    if longitud < 4:
        print(" Advertencia: La longitud debe ser de al menos 4 para asegurar una mezcla de caracteres.")
        return None

    # 1. Define el alfabeto de caracteres a usar
    # string.ascii_letters: a-z, A-Z
    # string.digits: 0-9
    # string.punctuation: !#$%&'()*+,-./:;<=>?@[\]^_`{|}~
    alfabeto = string.ascii_letters + string.digits + string.punctuation

    # 2. Asegúrate de que la contraseña contenga al menos un carácter de cada tipo
    # Esto garantiza una mayor complejidad y cumple con la mayoría de los requisitos de seguridad
    # Escogemos un carácter al azar de cada grupo
    contrasena = [
        secrets.choice(string.ascii_lowercase),  # Una minúscula
        secrets.choice(string.ascii_uppercase),  # Una mayúscula
        secrets.choice(string.digits),           # Un dígito
        secrets.choice(string.punctuation)       # Un símbolo
    ]

    # 3. Rellena el resto de la contraseña con caracteres aleatorios del alfabeto completo
    # La longitud restante es: longitud - 4 (por los 4 caracteres ya agregados)
    for i in range(longitud - 4):
        contrasena.append(secrets.choice(alfabeto))

    # 4. Mezcla la lista de caracteres para asegurar que los caracteres obligatorios
    # no aparezcan solo al principio (esto es crucial para la aleatoriedad).
```

```
    secrets.SystemRandom().shuffle(contrasena)

    # 5. Convierte la lista de caracteres en una cadena y la devuelve
    return "".join(contrasena)

if __name__ == "__main__":
    print("--- ? Generador de Contraseñas Seguras ---")

    # Pide al usuario la longitud de la contraseña
    while True:
        try:
            longitud_deseada = int(input("Introduce la longitud deseada para la contraseña (mínimo 4, ej: 16):
"))
            if longitud_deseada >= 4:
                break
            else:
                print("Por favor, introduce un número igual o mayor a 4.")
        except ValueError:
            print("Entrada no válida. Por favor, introduce un número entero.")

    # Genera y muestra la contraseña
    contrasena_generada = generar_contrasena_segura(longitud_deseada)

    if contrasena_generada:
        print("\n✅ Contraseña generada:")
        print(f"**{contrasena_generada}**")
        print("\n---")
        print(f"Longitud: {len(contrasena_generada)}")
        print("¡Recuerda almacenarla en un gestor de contraseñas seguro!")
```

Explicación del Código

1. **import secrets**: Este es el módulo crucial. A diferencia del módulo random, secrets está diseñado específicamente para generar números aleatorios criptográficamente seguros, lo cual es esencial para contraseñas y claves.
2. **import string**: Este módulo proporciona constantes de cadena útiles, como string.ascii_letters (todas las letras, mayúsculas y minúsculas), string.digits (0-9) y string.punctuation (símbolos).
3. **Definición del Alfabeto**: Se combinan las letras, dígitos y símbolos para crear el conjunto de caracteres (alfabeto) que puede usar la contraseña.
4. **Asegurar la Complejidad**: Los primeros cuatro caracteres se eligen de forma *obligatoria* (uno de minúsculas, mayúsculas, dígitos y símbolos). Esto garantiza que la contraseña cumpla con los requisitos mínimos de complejidad.
5. **Relleno Aleatorio**: El resto de la longitud se rellena con caracteres elegidos completamente al azar de todo el alfabeto.
6. **secrets.SystemRandom().shuffle(contrasena)**: Esta es la parte más importante después de la elección de caracteres. **Mezcla** toda la lista de caracteres generados. Si no se mezclara, la contraseña siempre comenzaría con una minúscula, mayúscula, dígito y símbolo, lo que reduciría su aleatoriedad y seguridad.
7. **"".join(contrasena)**: Convierte la lista de caracteres nuevamente en una cadena de texto para mostrarla.

¿Te gustaría que añadiera una opción para **generar múltiples contraseñas** a la vez o la opción de **excluir ciertos símbolos** (como comas o puntos) que a veces dan problemas?

¡**Sí**! Entiendo que quieres una extensión o modificación al código del **Generador de Contraseñas** que acabo de proporcionarte.

Aquí tienes el código de la aplicación mejorado para incluir la opción de **Generar Múltiples Contraseñas** y la función de **excluir ciertos símbolos** para mayor flexibilidad.

?□ﾇ Generador de Contraseñas con Opciones Avanzadas

Python

```
import secrets
import string

def generar_contrasena_segura(longitud=12, caracteres_excluidos=""):
    """
    Genera una contraseña aleatoria y segura de una longitud especificada,
    excluyendo un conjunto opcional de caracteres.

    Args:
        longitud (int): La longitud deseada para la contraseña. Por defecto es 12.
        caracteres_excluidos (str): Cadena de caracteres a excluir del alfabeto.

    Returns:
        str: La contraseña generada, o None si la longitud es muy pequeña.
    """
    if longitud < 4:
        # Longitud mínima forzada para asegurar al menos un carácter de cada tipo
        return None

    # 1. Define el alfabeto completo
    ALFABETO_COMPLETO = string.ascii_letters + string.digits + string.punctuation

    # 2. Crea el alfabeto final excluyendo los caracteres no deseados
    # Usamos un conjunto (set) para hacer la exclusión más eficiente
    alfabeto_final = "".join(c for c in ALFABETO_COMPLETO if c not in caracteres_excluidos)

    # Verifica que el alfabeto final aún contenga los tipos de caracteres necesarios
    if not any(c in alfabeto_final for c in string.ascii_lowercase):
        print("Error: El alfabeto no contiene minúsculas suficientes.")
        return None

    # 3. Asegura que la contraseña contenga al menos un carácter de cada tipo
    # (minúscula, mayúscula, dígito, símbolo) de los disponibles en el alfabeto final
    contrasena = []

    # Intenta obtener un carácter de cada tipo, si está disponible en el alfabeto final
    tipos_obligatorios = [
        [c for c in string.ascii_lowercase if c in alfabeto_final],
        [c for c in string.ascii_uppercase if c in alfabeto_final],
        [c for c in string.digits if c in alfabeto_final],
        [c for c in string.punctuation if c in alfabeto_final],
    ]

    for tipo in tipos_obligatorios:
        if tipo:
            contrasena.append(secrets.choice(tipo))
        # Si un tipo no está disponible, simplemente lo ignoramos, la contraseña será más débil.
```

```
    # 4. Rellena el resto de la contraseña
    # La longitud de los caracteres obligatorios puede ser menor a 4 si se excluyeron muchos caracteres
    caracteres_iniciales = len(contrasena)

    # Rellena hasta alcanzar la longitud deseada
    for _ in range(longitud - caracteres_iniciales):
        contrasena.append(secrets.choice(alfabeto_final))

    # 5. Mezcla los caracteres para una máxima aleatoriedad
    secrets.SystemRandom().shuffle(contrasena)

    # 6. Devuelve la contraseña como una cadena
    return "".join(contrasena)

def ejecutar_generador():
    """Función principal para interactuar con el usuario y generar las contraseñas."""
    print("--- ? Generador de Contraseñas (Avanzado) ---")

    # --- 1. Obtener la Longitud ---
    while True:
        try:
            longitud = int(input("Introduce la **longitud** deseada (mín. 4, ej: 16): "))
            if longitud >= 4:
                break
            else:
                print("La longitud debe ser de al menos 4 caracteres.")
        except ValueError:
            print("Entrada no válida. Por favor, introduce un número entero.")

    # --- 2. Obtener la Cantidad ---
    while True:
        try:
            cantidad = int(input("¿Cuántas contraseñas deseas generar? (ej: 3): "))
            if cantidad > 0:
                break
            else:
                print("Por favor, introduce un número positivo.")
        except ValueError:
            print("Entrada no válida. Por favor, introduce un número entero.")

    # --- 3. Obtener Caracteres a Excluir ---
    excluir = input(
        "Introduce **caracteres a excluir** (ej: i l 1 0 O , .). Deja vacío para no excluir ninguno: "
    )

    print("\nGenerando Contraseñas...")
    print("--- Resultados ---")

    # --- 4. Generar y Mostrar ---
    for i in range(cantidad):
        contrasena = generar_contrasena_segura(longitud, excluir)
        if contrasena:
            print(f"[{i+1}] {contrasena}")
        else:
```

```
                print(f"[{i+1}] No se pudo generar (longitud muy pequeña o caracteres excluidos inválidos).")

        print("------------------")


if __name__ == "__main__":
        ejecutar_generador()
```

Novedades y Mejoras

- **Generación Múltiple**: El script ahora solicita la **cantidad** de contraseñas a generar y utiliza un bucle for para llamar a la función tantas veces como se requiera.
- **Exclusión de Caracteres**:
- Se agregó un nuevo argumento caracteres_excluidos a la función principal.
- Esto es útil para excluir caracteres visualmente confusos como la 'l' minúscula, la 'I' mayúscula, el '1', la 'O' mayúscula o el '0'.
- La lógica ahora crea dinámicamente el alfabeto_final basándose en la lista de exclusión.
- **Manejo de Caracteres Obligatorios**: La función ahora es más robusta:
- Intenta asegurar al menos un carácter de cada tipo (**minúscula, mayúscula, dígito, símbolo**).
- Si el usuario excluye *todos* los caracteres de un tipo (por ejemplo, todos los dígitos), ese tipo simplemente no se incluirá, manteniendo la funcionalidad sin bloquear el programa.
  El siguiente video de YouTube muestra una explicación concisa sobre la implementación de la generación de contraseñas en Python utilizando el módulo secrets.
  [Genera Contraseñas Seguras con Una Línea de Código en Python]