UNIVERSIDAD INTERNACIONAL DEL ECUADOR UIDE

Facultad de Ciencias e Ingeniería

Carrera: Ingeniería en Software

Informe: Generador de Contraseñas

Asignatura: Lógica de Programación

Paralelo: 1-ECC-1F

Docente: **MONICA PATRICIA SALAZAR TAPIA**

Estudiante: Estuar Fernando Sigua Quimis

Fecha de entrega: 29 de junio de 2025

GENERADOR DE CONTRASEÑAS

Descripción General

Este programa genera contraseñas aleatorias personalizadas. Permite al usuario seleccionar si desea incluir letras mayúsculas, números y símbolos. La contraseña generada tendrá siempre 10 caracteres.

Objetivo: Facilitar la creación de contraseñas seguras ajustadas a preferencias del usuario.

Lenguaje: Python 3

CRONOGRAMA

Etapa	Duración estimada	Fechas
Análisis de requisitos	3 días	05 - 07 mayo
Diseño de algoritmo y diagramas	5 días	08 - 12 mayo
Desarrollo del generador (versión funcional)	6 días	13 - 18 mayo
Implementación de interfaz	6 días	19 - 24 mayo
Integración de funciones adicionales	5 días	25 - 29 mayo
Pruebas y corrección de errores	5 días	30 mayo - 03 junio
Optimización del código y limpieza general	4 días	04 - 07 junio
Revisión por parte de compañeros/profesor	3 días	08 - 10 junio
Documentación técnica y README	5 días	11 - 15 junio
Generación de diagramas finales	3 días	16 - 18 junio
Elaboración de presentación del proyecto	4 días	19 - 22 junio
Ensayo y revisión final del proyecto	5 días	23 - 27 junio
Entrega final del proyecto	1 día	29-jun

Funcionamiento

1. Solicita al usuario ingresar la longitud de la contraseña (solo acepta 10).
2. Verifica que el número ingresado sea correcto, repite la solicitud si es incorrecto.
3. Pregunta al usuario si quiere incluir:

   -Letras mayúsculas

-Números

-Símbolos

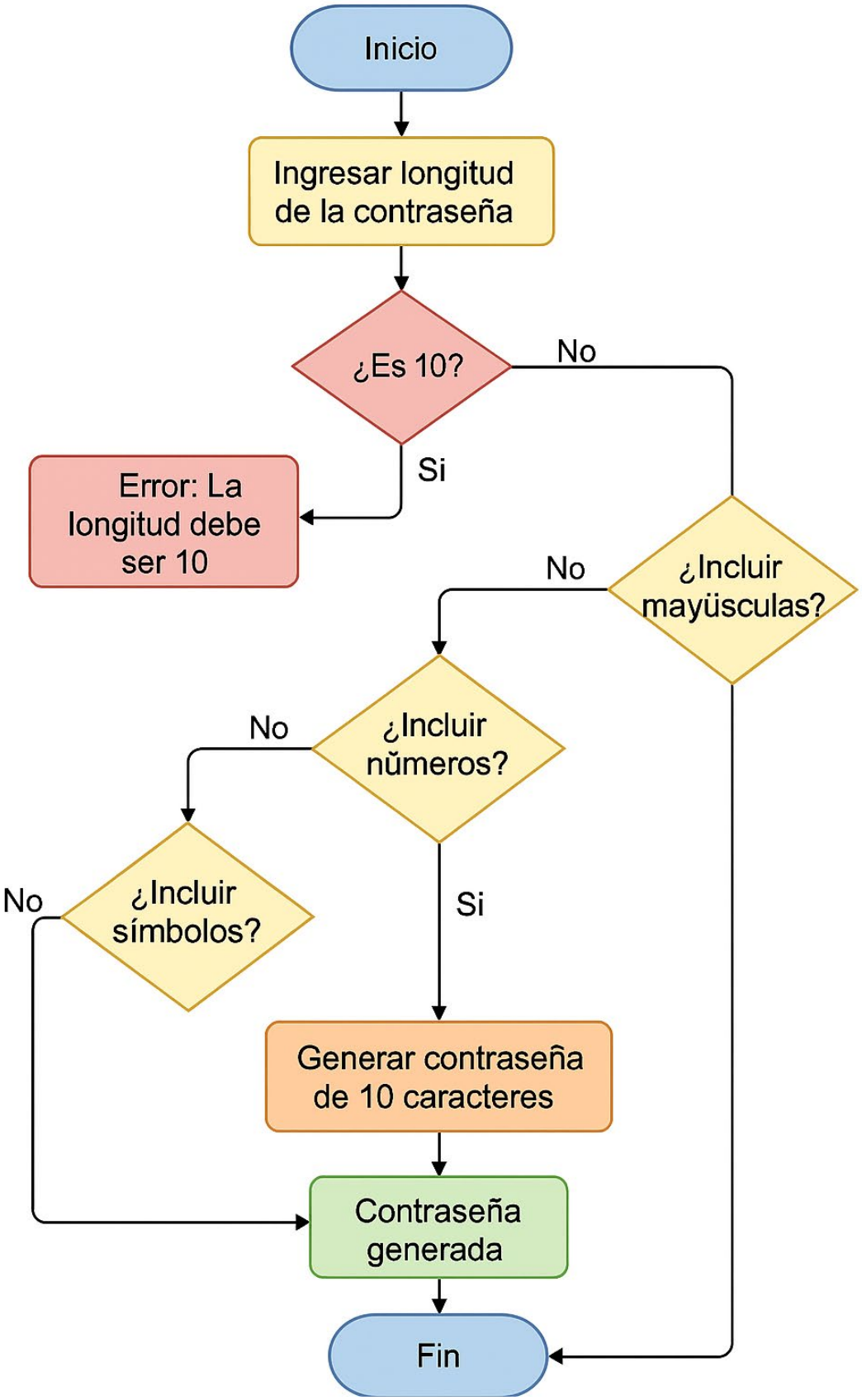
4. Genera una contraseña aleatoria mezclando las opciones elegidas.
5. Muestra la contraseña generada.

Diagrama de Flujo

Propósito del Diagrama

El diagrama de flujo representa gráficamente cómo funciona el programa paso a paso, permitiendo entender *qué hace* en cada momento:

1. Entrada del usuario (longitud de la contraseña)
2. Validación de que la longitud es correcta (10 caracteres)
3. Elección de características:
    - ¿Incluir mayúsculas?
    - ¿Incluir números?
    - ¿Incluir símbolos?
4. Proceso de generación aleatoria de la contraseña.
5. Salida: Mostrar la contraseña al usuario.


Inicio
Ingresar longitud de la contraseña
¿Es 10?
No
Si
Error: La longitud debe ser 10
No
¿Incluir mayüsculas?
No
¿Incluir nŭmeros?
No
¿Incluir símbolos?
Si
Generar contraseña de 10 caracteres
Contraseña generada
Fin

Información sobre el Código del Generador de Contraseñas

Propósito del Código

El código tiene como finalidad generar contraseñas aleatorias y seguras a partir de criterios personalizados que define el usuario:

- ¿Incluir mayúsculas?
- ¿Incluir números?
- ¿Incluir símbolos?

Funcionamiento del Programa Paso a Paso

Librerías Utilizadas

Librería	Propósito
random	Selección aleatoria de caracteres.
string	Acceso a letras, dígitos y símbolos predefinidos.

Definición de la función principal

def generar contraseña(longitud, incluir mayúsculas, incluir números, incluir símbolos):

Esta función recibe 4 parámetros:

-Longitud: cantidad de caracteres de la contraseña.

-Incluir mayúsculas: si debe incluir letras mayúsculas.

-Incluir números: si debe incluir números.

-Incluir símbolos: si debe incluir símbolos.

¿Qué hace?

-Genera una cadena de caracteres aleatorios según las opciones elegidas.

Selección de caracteres

```
caracteres = string.ascii_lowercase
if incluir_mayusculas:
    caracteres += string.ascii_uppercase
```

```
if incluir_numeros:
    caracteres += string.digits
if incluir_simbolos:
    caracteres += string.punctuation
```

- Siempre incluye minúsculas por defecto.
- Agrega otros tipos de caracteres según las respuestas del usuario.

Generación de contraseña

```
contraseña = ''.join(random.choice(caracteres) for _ in range(longitud))
```

- Utiliza random.choice() para seleccionar al azar cada carácter de la contraseña final.

Validación de la longitud

```
while True:
    entrada = input(...)
```

- El bucle while garantiza que el usuario introduzca exactamente 10 caracteres, repitiendo la solicitud si es incorrecto.

Interacción con el usuario

```
usar_mayus = input("¿Incluir mayúsculas? (s/n): ").lower() == 's'
```

- Pregunta al usuario si desea incluir diferentes tipos de caracteres.

Ejemplo de uso

Ingrese la longitud de la contraseña (debe ser 10): 10

¿Incluir mayúsculas? (s/n): s

¿Incluir números? (s/n): s

¿Incluir símbolos? (s/n): n

Contraseña generada: aN3vplKpeW

Posibles Mejoras

- Permitir longitudes personalizables (ahora está fija en 10).
- Añadir verificación para asegurar al menos un carácter de cada tipo seleccionado.
- Incorporar una interfaz gráfica para facilitar su uso.
- Guardar contraseñas generadas en un archivo si se desea.

ACTUALIZACIONES FINALES DE ESTE PROYECTO CON UN NUEVO CODIGO

Código Original (simplificado)

- Parámetros de función: 4 (longitud, y tres booleanos para incluir mayúsculas, números y símbolos)
- Construcción de caracteres con concatenación de strings (+=).
- Solicitud simple de longitud sin validar más que el número.
- Pedir opciones con inputs directos y comparar == 's' sin validación estricta.
- Genera contraseña directamente y la muestra.

CAMBIOS IMPLEMENTADOS

Validación estricta para la longitud fija:

while longitud! = LONGITUD_REQUERIDA:

- Ahora solo acepta la longitud exacta (p. ej. 16) y **repregunta** hasta que el usuario ponga ese valor.
- En el código original solo imprimía el error, pero el bucle no era tan claro para repetir la pregunta.

Validación para las opciones (sí/no):

- En el código actualizado definiste esta función para validar que solo se responda con 's' o 'n' y volver a pedir en caso contrario:

```
def pedir_opcion(mensaje):
  while True:
    respuesta = input(mensaje).strip().lower()
    if respuesta == 's':
      return True
    elif respuesta == 'n':
      return False
    else:
      print("❌ Error: Responde solo con 's' (sí) o 'n' (no).")
```

- Esto evita que el usuario introduzca cualquier otro carácter, cosa que en el código antiguo no había.

Uso de diccionarios para manejar opciones:

```
opciones_usuario = {
    "mayusculas": pedir_opcion("¿Incluir mayúsculas? (s/n): "),
    "numeros": pedir_opcion("¿Incluir números? (s/n): "),
    "simbolos": pedir_opcion("¿Incluir símbolos? (s/n): ")
}
```

- Esto te permite agrupar las opciones en un solo objeto, y pasar a la función generar_contraseña un único parámetro (diccionario).
- Más organizado y escalable que usar múltiples parámetros booleanos.

Uso de listas para almacenar los caracteres permitidos:

- En el código actualizado, usas listas y extend en lugar de concatenar strings:

```
caracteres_permitidos = tipos_caracteres["minusculas"][:]
if opciones["mayusculas"]:
    caracteres_permitidos.extend(tipos_caracteres["mayusculas"])
```

- Esto es mejor para manipular colecciones, y luego eliges aleatoriamente caracteres de la lista.
- Aunque en el código antiguo se usaban strings concatenados y funcionaba, usar listas es más flexible.

Parámetro fijo de longitud definido con constante (p. ej. 16):

- En el código actualizado tienes:

```
LONGITUD_REQUERIDA = 16
```

- En el antiguo, la longitud era 10.

- Así se puedes modificar fácilmente la longitud sin cambiar todo el código.

---

6. **Separación de funciones para mayor claridad:**

- En el código actualizado, tienes la función pedir_opcion() para validar opciones, que no existía antes.
- También generar_contraseña() está diseñada para aceptar un diccionario de opciones, no solo booleanos separados.

Conclusión

El código original para generar contraseñas cumplía con la función básica de crear contraseñas con opciones para mayúsculas, números y símbolos, sin embargo, carecía de validaciones robustas y una estructura modular eficiente. No se validaban correctamente las entradas del usuario, lo que podía generar errores o comportamientos inesperados, y el manejo de caracteres se hacía mediante concatenación de strings, lo cual es menos flexible.

Las mejoras implementadas en el código actualizado incluyen:

- Validación estricta y repetitiva para asegurar que la longitud de la contraseña sea la requerida, mejorando la experiencia del usuario.
- Una función dedicada para validar respuestas sí/no, asegurando que solo se acepten respuestas válidas y evitando errores.
- Uso de un diccionario para agrupar las opciones del usuario, lo que facilita la extensión y mantenimiento del código.
- Manejo de los caracteres permitidos mediante listas, lo que mejora la flexibilidad y claridad del código.
- Definición de constantes para parámetros clave como la longitud, facilitando futuros ajustes.
- Modularización y organización del código para mayor claridad y reutilización.