# **Pistas y Guía para el Ejercicio Integrador**

Esta guía fue diseñada para asistirlo en la finalización del ejercicio integrador. Proporciona, paso a paso, pistas para cada sección y consejos útiles para la depuración de su código.

## **Pistas para la Función analizar\_texto()**

### **TODO 1: Crear instancia de ProcesadorTexto**

*Pista: Así se crea un objeto a partir de una clase.*

# 'texto' es el parámetro que recibe la función

procesador = ProcesadorTexto(texto)

### **TODO 2: Obtener métricas**

*Pista: Los métodos de la clase le dan acceso a las métricas.*

total\_palabras = procesador.contar\_palabras()

palabras\_unicas = procesador.contar\_palabras\_unicas()

frecuencias = procesador.calcular\_frecuencia()

### **TODO 3: Encontrar palabra más frecuente (Opcional)**

*Pista: Para encontrar la clave con el valor máximo en un diccionario, puede utilizar la función max() con el argumento key.*

if frecuencias: # Verifica que el diccionario no esté vacío

palabra\_mas\_frecuente = max(frecuencias, key=frecuencias.get)

repeticiones = frecuencias[palabra\_mas\_frecuente]

else:

palabra\_mas\_frecuente = "N/A"

repeticiones = 0

### **TODO 4: Formatear el resumen**

*Pista: Utilice f-strings para crear un texto multilínea con las métricas de forma clara y ordenada.*

resumen = (

f"--- Resumen del Análisis ---\n"

f"Total de Palabras: {total\_palabras}\n"

f"Palabras Únicas: {palabras\_unicas}\n"

f"Palabra Más Frecuente: '{palabra\_mas\_frecuente}' ({repeticiones} repeticiones)"

)

### **TODO 5: Devolver resultados**

*Pista: Una función en Python puede devolver múltiples valores separados por comas. Gradio los recibirá en el orden correspondiente.*

return resumen, frecuencias

## **Pistas para la Interfaz de Gradio**

### **TODO 6: Configurar la función**

*Pista: Al parámetro fn se le asigna el nombre de la función que usted creó, sin utilizar paréntesis.*

fn=analizar\_texto,

### **TODO 7: Configurar las salidas**

*Pista: Gradio posee componentes específicos para diferentes tipos de datos. Utilice gr.Textbox para el resumen y gr.JSON para el diccionario de frecuencias.*

outputs=[

gr.Textbox(label="Resumen del Análisis"),

gr.JSON(label="Frecuencia de Palabras")

],

### **TODO 8: Lanzar la aplicación**

*Pista: El método .launch() es el que inicia la interfaz web. El uso de share=True genera un enlace público temporal para que usted pueda compartir su aplicación.*

demo.launch(share=True)

## **🔍 Consejos para Depurar**

* **Prueba por partes**: Antes de conectar todo con Gradio, asegúrese de que su función analizar\_texto() funcione correctamente por sí sola. Ingrésele un texto de ejemplo y compruebe que los resultados sean los esperados.
* **Uso de print()**: Si algo no funciona, agregue print() temporales en su función para observar los valores de las variables en cada paso. Por ejemplo: print(f"Total palabras: {total\_palabras}").
* **Lectura de errores**: Si su aplicación falla, Gradio mostrará mensajes de error en rojo en la consola o en la interfaz. Léalos con atención, ya que suelen dar pistas muy claras sobre el problema.
* **Reinicio del entorno**: En ocasiones, los problemas extraños se solucionan al reiniciar el *runtime* de Colab (**Runtime → Restart runtime**).

## **💡 Extensión Opcional para Estudiantes Avanzados**

Si usted finaliza con antelación y desea un desafío adicional, puede mejorar su analizador al añadirle estas funcionalidades:

* **Longitud media de palabras**:

# Añada este método a su clase ProcesadorTexto

def longitud\_media\_palabras(self):

if not self.palabras:

return 0

total\_letras = sum(len(palabra) for palabra in self.palabras)

return total\_letras / len(self.palabras)

* **Palabras más frecuentes (top 5)**:

# Para obtener las 5 palabras más frecuentes

from collections import Counter

top\_5 = Counter(frecuencias).most\_common(5)

* **Diversidad léxica**:

# Calcula qué porcentaje de palabras son únicas

diversidad = (palabras\_unicas / total\_palabras) \* 100 if total\_palabras > 0 else 0

## **📝 Ejemplo de Texto para Probar**

Puede utilizar este texto para verificar que su aplicación funciona correctamente:

Python es un lenguaje de programación versátil y poderoso. Python se utiliza en muchas áreas, desde desarrollo web hasta ciencia de datos. La simplicidad de Python lo hace ideal para principiantes, pero su potencia satisface incluso a programadores expertos.

Se recuerda que la experimentación y los errores constituyen una parte fundamental del aprendizaje. No debe preocuparse si la aplicación no funciona en el primer intento. Es importante perseverar.