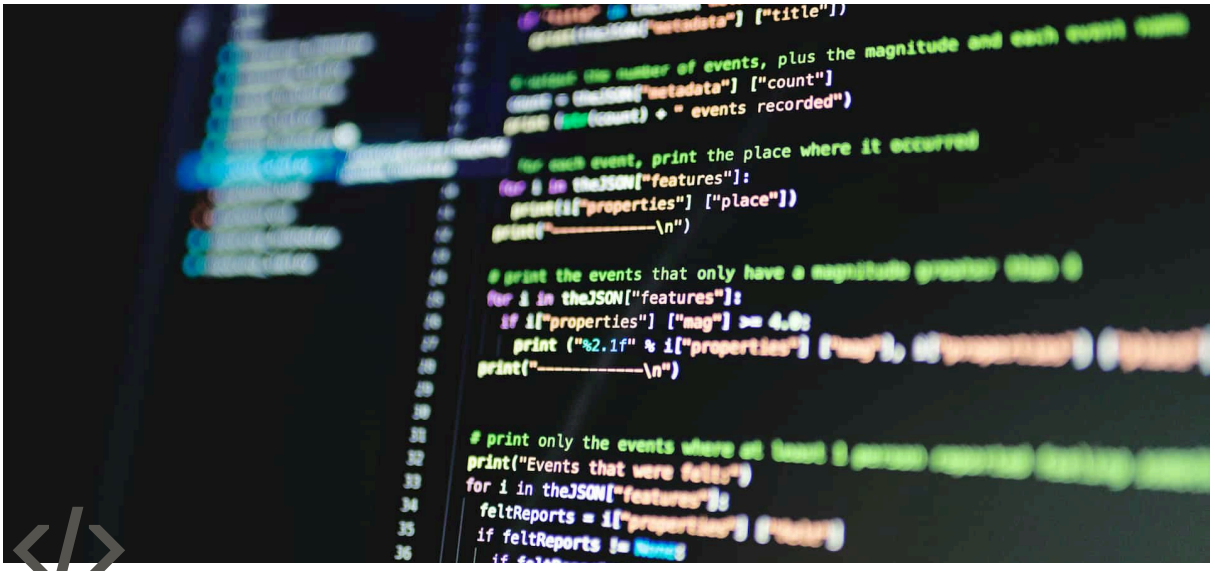

Documentación ejercicio 4.

💡 **Problema #4:**

Se desea implementar el historial de acciones realizadas por un usuario en un editor de texto (como escribir, borrar, pegar, copiar). Cada acción debe guardarse en orden y poder recorrerlas en ambas direcciones, simulando las acciones de Deshacer y Rehacer.

Editor_De_Texto.py:

```
from colorama import Fore, Back, Style
```

Se importa la librería “colorama” para agregar colores a los mensajes que se van a imprimir en consola.

```
class Nodo:

def __init__(self, description):
```

```
self.description = description

self.prev = None

self.next = None
```

Se crea el Nodo con el argumento “description” para guardar en esa variable el texto o acción realizada.

- prev apunta al nodo anterior, mientras que next al siguiente.

Esto es necesario para poder deshacer y rehacer acciones de forma dinámica.

```
class Historial:

def __init__(self):

self.current = None
```

Current es una referencia al nodo actual en el historial. Cuando el historial está vacío o recién creado, current es None, lo que significa que no hay acciones registradas todavía.

```
def crearNodo(self, description):

# Crear y conectar un nuevo "Nodo"

new_action = Nodo(description)

# Si hay un nodo actual, conectar el nuevo nodo al anterior

if self.current:

self.current.next = new_action

new_action.prev = self.current
```

```
self.cuerrent = nw_action
```

Se le pasa como argumento “self” para referirse a la instancia actual, y “description” para el texto que queremos guardar.

El método crea un nuevo nodo con “description.”

Si ya existe un current (ya hay acciones hechas):

- Conecta el nuevo nodo como siguiente del current.
- Conecta el current como anterior del nuevo nodo.

Luego, el nuevo nodo pasa a ser el nuevo current, porque es la acción más reciente.

```
def deshacer(self):

# Moverse a prev

#Si hay un nodo actual y un nodo anterior, moverse al anterior

if self.current and self.current.prev:

print(Fore.GREEN+"\nDeshaciendo acción:", self.current.description)

self.current = self.current.prev

else:

print(Fore.YELLOW+"\nNo hay más acciones para deshacer.")
```

Def deshacer(self): Define el método deshacer dentro de la clase Historial.

Con “self.current”, se verifica si hay un nodo actual, es decir, si no está vacío el historial.

Con “self.current.prev”, se verifica si el nodo actual tiene nodo anterior (es decir, que no sea el primer nodo)

Si ambas condiciones son verdaderas, se pasa al siguiente bloque de código. Donde “self.current.prev” apunta al nodo anterior al actual.

Se actualiza “self.current” para que apunte al nodo anterior (retrocediendo en el historial).

Else si no hay nodo anterior o si el historial está vacío (“self.current” o “self.current.prev” = None).

```
def rehacer(self):

# Moverse a next

#Si hay un nodo actual y un nodo siguiente, moverse al siguiente

if self.current and self.current.next:

print(Fore.GREEN+"\nRehaciendo acción:", self.current.description)

self.current = self.current.next

else:

print(Fore.YELLOW+"No hay más acciones para rehacer.")
```

Funciona de manera similar al método deshacer, pero en dirección contraria.

- “self.current.next” devuelve el nodo siguiente al actual (la acción que viene después).
- “self.current = self.current.next” mueve el current hacia adelante, hacia la acción deshecha.

```
def mostrarHistorial(self):

# Mostrar lista

current = self.current
```

```
while current:

print(current.description)

current = current.prev

print(Style.BRIGHT+"Fin del historial")
```

Con “current = self.current”, se asigna al nodo actual (último nodo agregado al historial). Esto es porque el historial se recorre hacia atrás desde el nodo actual.

Se pasa al while, que se ejecutará mientras haya nodos en el historial. Comienza con el nodo actual y va hacia atrás (current.prev).

En cada iteración del bucle, se imprime la descripción de la acción almacenada en el nodo actual.

Se utiliza “current = current.prev” para después de imprimir la descripción, se mueva al nodo anterior (prev) para seguir mostrando las acciones anteriores.

```
def mostrar_Texto(self):

if self.current:

print("\n--- Documento Actual ---")

contenido_limpio = self.current.description.replace(";", "\n")

print(self.current.description)

print("-------------------------\n")

else:

print(Fore.YELLOW+"\nDocumento vacío.\n")

return
```

```
# Si no hay nodos (el historial está vacío), mostrar mensaje
```

“If self.current” verifica si hay un nodo actual. Si self.current es None, significa que no hay acciones en el historial.

contenido_limpio = self.current.description.replace(";", "\n"): Se reemplazan todos los puntos y coma por saltos de línea. Esto se hace para formatear el contenido del texto de manera que cada acción registrada se muestre en una nueva línea, pero no se usa en el print, ya que “self.current.description” se imprime directamente.

```
def mostrar_Texto_Sin_Formato(self):

if self.current:

contenido_limpio = self.current.description.replace(";", "\n")

print(self.current.description)

else:

print(Fore.LIGHTRED_EX+"\nDocumento vacío.\n")
```

Muestra el contenido de la acción actual sin ningún formato especial (mantiene el texto tal como está).

Main.py:

```
from Editor_De_Texto import Historial
```

Importa la clase Historial desde el archivo Editor_De_Texto.

```
import os

from colorama import init, Fore, Style
```

```
init(autoreset=True)
```

Inicialización de Colorama.

```
def clear():

# Limpiar la consola

os.system('clear')

os.system('cls')

def contar_palabras(texto):

palabras = texto.replace("\n", " ").split()

return len(palabras)
```

El método “texto.replace("\n", " ")” reemplaza todos los salto de línea con un espacio en blanco, y el método “split()”, se utiliza para dividir el texto en palabras. Este método divide la cadena en una lista de palabras separadas por espacios. Si hay múltiples espacios consecutivos, split() los ignora y divide solo por los espacios.

Se utiliza la función “len()” para contar el número de elementos dentro de la lista.

```
def main():

editorHistorial = Historial()

editorHistorial.crearNodo("") #Acá creamos un nodo vacío
```

“editorHistorial = Historial()” Crea una instancia de la clase Historial, que gestionará el historial de las acciones realizadas en el editor de texto.

“editorHistorial.crearNodo("")” Crea el primer nodo vacío en el historial. Esto inicializa la lista de historial.

```
lineas_maximas = 10
```

Número máximo de líneas = 10

```
while True:

clear()

print('=' * 35)

print("Editor de texto")

print('=' * 35)

print("1. Escribir Texto")

print("2. Deshacer")

print("3. Rehacer")

print("4. Mostrar Documento")

print("5. Salir")

print('=' * 35)

try:

opc = int(input("Seleccione una opción: "))

except ValueError:

print(Fore.RED+"Ingrese un número válido.\n")

continue
```

Se valida que sea un número, en caso contrario, se vuelve a pedir el valor.

```
match opc:

# Escribir texto

case 1:

clear()

# Obtenemos el texto actual del documento

texto_actual = editorHistorial.current.description

# Pedimos al usuario que escriba nuevo texto

nuevo_texto = input("Escribe el texto que quieres agregar: ")

# Procesar si hay punto y coma para hacer saltos de línea

nuevo_texto = nuevo_texto.replace(";", "\n")

# Concatenar el texto nuevo

if len(texto_actual) > 50:

texto_actualizado = texto_actual + "\n" + nuevo_texto

else:

texto_actualizado = texto_actual + " " + nuevo_texto

# Separar líneas

lineas = texto_actualizado.split("\n")

if len(lineas) <= lineas_maximas:
```

```
# Si no supera el límite, crear un solo nodo

editorHistorial.crearNodo(texto_actualizado)

else:

# Si supera el límite, dividir en páginas

pagina1 = "\n".join(lineas[:lineas_maximas])

pagina2 = "\n".join(lineas[lineas_maximas:])

editorHistorial.crearNodo(pagina1)

editorHistorial.crearNodo("---------- Nueva Página ----------\n" + pagina2)

# Mostrar el texto actualizado

print(Fore.GREEN+"\nTexto actualizado:\n")

editorHistorial.mostrar_Texto_Sin_Formato()

# Mostrar contador de palabras

cantidad_palabras = contar_palabras(texto_actualizado)

print("\n---------------------------------------")

print(f"Número de palabras actuales: {cantidad_palabras}")

print("---------------------------------------\n")

input(Fore.YELLOW+"Presiona Enter para continuar...")
```

Luego de que el usuario ingresa un nuevo texto, se actualiza el contenido combinándolo con el texto que ya existía. Se hace una validación:

Si el texto anterior tenía más de **50 caracteres (len(texto_actual) > 50)**, entonces se concatena el nuevo texto en una nueva línea (\n).

Si no supera los **50 caracteres,** se agrega el nuevo texto separado por un espacio.

Si el usuario escribe: **“;”** al final de la palabra, crea un salto de linea

Después, con **“texto_actualizado.split("\n")”**, se separa el texto en líneas, utilizando el salto de línea como delimitador. Esto convierte el texto en una lista de líneas individuales.

Luego se analiza cuántas líneas resultaron:

Si la cantidad de líneas es menor o igual a **lineas_maximas (10 líneas)**, se guarda todo el texto en un solo nodo del historial usando editorHistorial.crearNodo(texto_actualizado).

Si supera las **10** líneas, entonces el texto se divide en dos partes:

pagina #1 contiene las primeras 10 líneas.

pagina #2 contiene el resto de las líneas.

Una vez actualizado el historial, se imprime el texto actualizado usando el método “mostrar_Texto_Sin_Formato()”, que muestra el contenido tal cual se guardó en el nodo.

Luego, se calcula cuántas palabras tiene el texto actualizado:

Se usa la función “contar_palabras(texto_actualizado)”, que reemplaza los saltos de línea por espacios y divide el texto en palabras para contar cuántas hay.

```
# Deshacer

case 2:

editorHistorial.deshacer()

editorHistorial.mostrar_Texto()

input(Fore.YELLOW+"Presiona Enter para continuar...")
```

editorHistorial.deshacer(): Llama al método deshacer del historial para revertir la última acción.

editorHistorial.mostrar_Texto(): Muestra el texto después de deshacer la acción.

```
# Rehacer

case 3:

editorHistorial.rehacer()

editorHistorial.mostrar_Texto()

input(Fore.YELLOW+"Presiona Enter para continuar...")
```

editorHistorial.rehacer(): Llama al método rehacer para reejecutar la acción que fue deshecha.

editorHistorial.mostrar_Texto(): Muestra el texto después de rehacer la acción.

```
# Mostrar documento

case 4:

#editorHistorial.mostrarHistorial()

editorHistorial.mostrar_Texto()

input(Fore.YELLOW+"Presiona Enter para continuar...")
```

editorHistorial.mostrar_Texto(): Muestra el texto actual del documento.

```
# Salir

case 5:

print(Fore.CYAN+"\nSaliendo del editor de texto.")

print(Fore.CYAN+"Gracias por usar el programa. ¡Hasta luego! 👋")
```

```
break

case _:

print(Fore.RED+"Opción no válida. Intenta de nuevo.")

input(Fore.YELLOW+"Presiona Enter para continuar...")

if __name__ == "__main__":

main()
```

Se ejecuta la función main() solamente si ese archivo se está corriendo directamente. Así, si alguien importa este archivo como módulo en otro programa, el main() no se ejecutará automáticamente.