Universidad Americana

Prueba Semana 10

Integrantes:

Est. Yassly Diaz

Est. Paulo Solis

Est.Walter Morales

Est. Julio Salamanca

Est. Erick Guido

Docente:

Ing. Silvia Ticay

Fecha: 26/05/2025

Evaluación práctica estructura de datos lineales

Ejercicio: Separación de elementos usando Pila y Cola

Objetivo:
Dado una cola con elementos y una pila vacía, desarrollar un programa que procese la cola de forma que:

Los elementos que se encuentran en posiciones pares (0, 2, 4, ...) permanezcan en la cola.

Los elementos que se encuentran en posiciones impares (1, 3, 5, ...) se transfieran a la pila.

Consideraciones:

La posición de los elementos se cuenta a partir de cero (la primera posición es la 0, la segunda es la 1, etc.).

La operación debe realizarse recorriendo la cola una sola vez.

Al finalizar, la pila contendrá los elementos impares en orden LIFO y la cola solo los elementos pares en su orden original.

Ejemplo:

Cola original: [A, B, C, D, E]
Resultado:

Cola: [A, C, E]

Pila: ['D', 'B'] (último en entrar, primero en salir)

- Actividad en grupo.
- Implementar módulos y conceptos básicos de POO.
- Entregar la documentación del código en un documento.
- Subir enlace de repositorio en GitHub.
- **Valor: 20 pts.**

Documentación:

CLASE ESTRUCTURA:

- Clase Nodo: Elemento básico para pilas y colas
- Clase Cola: Cola vacía con punteros frente y final
- Método enqueue: Agregar elemento al final de la cola
- Método dequeue: Eliminar el primer elemento de la cola
- Método imprimir (Cola): Mostrar todos los elementos
- Clase Pila: Pila vacía con puntero al tope
- Método pop: Eliminar y devolver el tope de la pila
- Método imprimir (Pila): Mostrar todos los elementos desde el tope

Clase Nodo: Elemento básico para pilas y colas

La clase `Nodo` sirve para representar un solo elemento dentro de estructuras como pilas o colas. Cada nodo guarda dos cosas: un **dato**, que puede ser cualquier valor (como un número o una palabra), y una **referencia al siguiente nodo**, que al principio es `None` porque aún no está conectado a otro. Cuando se crean varios nodos y se enlazan entre sí, esta estructura permite organizarlos en secuencia, como si fueran eslabones de una cadena.

```
class Nodo:
    def __init__(self, dato):
        self._dato = dato
        self._siguiente = None
```

Clase Cola: Cola vacía con punteros frente y final

La clase `Cola` crea una cola vacía. Usa dos variables: `frente` (el primer elemento) y `final` (el último). Al inicio, ambas están en `None` porque la cola no tiene datos aún.

```python
class Cola:
    def __init__(self):
        self.frente = None  # Inicio de la cola
        self.final = None   # Final de la cola
```

Método enqueue: Agregar elemento al final de la cola

El método `enqueue` se encarga de agregar un nuevo elemento al final de la cola. Primero crea un nodo con el dato que se quiere insertar. Si la cola está vacía, ese nodo se convierte tanto en el frente como en el final. Si ya hay elementos, el nuevo nodo se conecta al final de la cola y luego se actualiza el puntero `final` para que apunte al nuevo nodo, manteniendo así el orden de ingreso.

```python
# Agrega un elemento a la cola
def enqueue(self, dato):
    nuevo_nodo = Nodo(dato)
    if self.final is None:
        self.frente = nuevo_nodo
        self.final = nuevo_nodo
    else:
        self.final._siguiente = nuevo_nodo
        self.final = nuevo_nodo
```

Método dequeue: Eliminar el primer elemento de la cola

El método `dequeue` elimina el primer elemento de la cola. Si la cola está vacía, muestra un mensaje de error y devuelve `None`. Si hay elementos, guarda el dato del nodo del frente, luego mueve el puntero `frente` al siguiente nodo. Si al eliminar el nodo la cola queda vacía, también se actualiza `final` a `None`. Finalmente, devuelve el dato eliminado.

```
# Elimina un elemento de la cola
def dequeue(self):
    if self.frente is None:
        print("Error: cola vacía. No se puede eliminar.")
        return None
    eliminado = self.frente._dato
    self.frente = self.frente._siguiente
    if self.frente is None:
        self.final = None
    return eliminado
```

Método imprimir (Cola): Mostrar todos los elementos

El método `imprimir` muestra todos los elementos de la cola desde el frente hasta el final. Si la cola está vacía, imprime un mensaje indicando eso. Si tiene elementos, recorre cada nodo uno por uno y muestra su dato en pantalla.

```
# Método para imprimir la cola
def imprimir(self):
    actual = self.frente
    if actual is None:
        print("La cola está vacía.")
    else:
        while actual is not None:
            print(actual._dato)
            actual = actual._siguiente
```

Clase Pila: Pila vacía con puntero al tope

La clase `Pila` representa una pila vacía inicialmente, usando el atributo `_tope` para indicar el elemento que está arriba. El método `push` agrega un nuevo dato a la pila creando un nodo y colocándolo encima del tope actual, actualizando `_tope` para que apunte a ese nuevo nodo. Así, el último elemento insertado queda siempre arriba.

```
class Pila:
    def __init__(self):
        self._tope = None  # Inicia vacía

    def push(self, dato):
        nuevo_nodo = Nodo(dato)
        nuevo_nodo._siguiente = self._tope
        self._tope = nuevo_nodo
        print(f"Elemento '{dato}' insertado.")
```

Método pop: Eliminar y devolver el tope de la pila

El método `pop` elimina y devuelve el elemento que está en el tope de la pila. Si la pila está vacía, muestra un mensaje de error y devuelve `None`. Si hay elementos, guarda el dato del nodo en el tope, luego actualiza `_tope` para que apunte al siguiente nodo, removiendo así el elemento superior.

```
  def pop(self):
      if self._tope is None:
          print("Error: pila vacía. No se puede
          return None
      eliminado = self._tope._dato
      self._tope = self._tope._siguiente
      return eliminado
```

Método imprimir (Pila): Mostrar todos los elementos desde el tope

El método `imprimir` muestra todos los elementos de la pila desde el tope hasta el fondo. Recorre cada nodo empezando por `_tope` y va imprimiendo sus datos en una misma línea separados por espacios, hasta llegar al final de la pila. Luego añade un salto de línea.

```
def imprimir(self):
    actual = self._tope
    while actual:
        print(actual._dato, end=" ")
        actual = actual._siguiente
    print()
```

CLASE PROCESADOR

```
from ClaseEstrutura import Cola, Pila
```

Esta línea importa las clases “Pila” y “Cola” desde el módulo “ClaseEstructura”.

```
def procesar_cola_y_pila(cola):
```

Esta línea define una función llamada “procesar_cola_y_pila”, esta toma como parametro una instancia de “cola”.

```
pila = Pila()
contador = 0
```

Se crea una nueva instancia de “Pila”, esta se usará para almacenar los datos impares. Se crea un contador y se establece como 0. Se usará para llevar un seguimiento de los elementos extraídos de la cola.

```
while not cola.is_empty():
    elemento = cola.dequeue()
```

Este while se repite siempre y cuando la cola esté vacía.
“cola.dequeue” extra el elemento al frente de la cola.

```
    if contador % 2 == 0:  # Posiciones pares
        cola.enqueue(elemento)
    else:  # Posiciones impares
        pila.push(elemento)
    contador += 1
```

Si la posición del elemento (contador) es par, se reinyecta en la cola (enqueue)
Si es impar, se agrega a la pila (push).
Luego se incrementa el contador para pasar al siguiente indice.

```
return pila
```

La función devuelve la pila resultante.

Main

- OBJETIVO DEL CÓDIGO PRINCIPAL

```
from Modulos.ClaseEstrutura import Cola, Pila
```

Se importan las clases Cola y Pila desde un archivo externo (modularización del código).

Este archivo debe estar ubicado en una carpeta llamada Modulos, y debe llamarse ClaseEstrutura.py.

- ESTRUCTURA DEL PROGRAMA

```
if __name__ == "__main__":
```

Esta línea se asegura de que el código dentro de este bloque solo se ejecute si corres el archivo directamente (no si lo importas desde otro).

- INSTANCIACIÓN DE LAS ESTRUCTURAS

```
mi_pila = Pila()
mi_cola = Cola()
```

`mi_pila`: es una pila (estructura LIFO: último en entrar, primero en salir).

`mi_cola`: es una cola (estructura FIFO: primero en entrar, primero en salir).

- PROCESAMIENTO DE UNA LISTA DE NÚMEROS

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
for numero in numeros:
    if numero % 2 == 0:
        mi_cola.enqueue(numero)
    else:
        mi_pila.push(numero)
```

Lista simple de enteros del 1 al 10.

- MOSTRAR RESULTADOS

```
print("Números en la cola (pares):")
mi_cola.imprimir()

print("Números en la pila (impares):")
mi_pila.imprimir()
```

-Imprime los elementos almacenados en la cola y la pila.
-Se asume que las clases `Cola` y `Pila` tienen un método `imprimir()` que muestra su contenido de forma ordenada.