UNIVERSIDAD AMERICANA

Documentacion de Codigo

Docente:

- Silvida gigdali Ticay Lopez

Estudiantes:

- Diego Armando Urbina Avilés
- Julio Cesar Delgadillo Pineda

Fecha: 5/26/2025

Explicacion de pila.py

La clase `Pila` implementa una estructura LIFO.

En condigo

```
class Pila:
    def __init__(self):
        self._items = []
```

Este método inicializa la lista interna vacía que almacena los elementos de la pila. Usamos una lista de Python para aprovechar las operaciones O(1) de append y pop.

Método `push` y `pop`:

```
    def push(self, item):
        self._items.append(item)


    def pop(self):
        return self._items.pop()
```

- `push(item)`: Agrega `item` al final de la lista, que representa el tope de la pila.
- `pop()`: Elimina y devuelve el último elemento (tope), lanzando `IndexError` si la pila está vacía.

Explicacion de cola.py

La clase `Cola` implementa una estructura FIFO usando `collections.deque` para eficiencia.

Creación e inserción:

```
from collections import deque

class Cola:
    def __init__(self, iterable=None):
        self._items = deque(iterable) if iterable else deque()
    def enqueue(self, item):
        self._items.append(item)
```

- `__init__`: Si se suministra un iterable, inicializa la cola con sus elementos, sino inicia vacía.
- `enqueue(item)`: Añade `item` al final de la `deque`, operación amortizada O(1).

Extracción de elementos:

```
    def dequeue(self):
        return self._items.popleft()
```

- `dequeue()`: Extrae el primer elemento de la cola (frente) en O(1). Lanza `IndexError` si la cola está vacía.

Explicacion de separador.py

La clase `Procesador` contiene lógicas de separación implementadas con un solo recorrido de la cola.

Separación por posición:

```
class Procesador:
    @staticmethod
    def separar_por_posicion(cola, pila):
        n = len(cola)
        for i in range(n):
            x = cola.dequeue()
            if i % 2 == 0:
                cola.enqueue(x)
            else:
                pila.push(x)
```

Iteramos exactamente `n` veces, donde `n` es el tamaño inicial de la cola. Índices pares reencolan el elemento; impares lo envían a la pila.

Separación por valor numérico:

```
    @staticmethod
    def separar_por_valor(cola, pila):
        n = len(cola)
        for _ in range(n):
            x = cola.dequeue()
            try:
                num = int(x)
                if num % 2 == 0:
                    cola.enqueue(x)
                else:
                    pila.push(x)
            except ValueError:
                cola.enqueue(x)
```

Intentamos convertir cada elemento a entero. Si es par, reencolamos; si es impar, apilamos. Cualquier elemento no convertible permanece en la cola.

Explicacion del main.py

El script principal ofrece al usuario: (1) usar datos de ejemplo o ingresar los suyos, (2) seleccionar el modo de separación.

Selector de datos y modo:

```
elems = ejemplo if usar_ejemplo else obtener_elementos_usuario()
modo = input('1: posición, 2: valor numérico')
if modo == '2':
    Procesador.separar_por_valor(cola, pila)
else:
    Procesador.separar_por_posicion(cola, pila)
```

Según la elección (opción `2` o cualquier otro valor), se invoca el método correspondiente de `Procesador`.