Trabajo: Listas Enlazadas y Pilas

# Ejercicio 1: Sistema para el Control de Animales en un Zoológico

Objetivo: Diseñar e implementar un sistema básico para controlar el registro de animales en un zoológico utilizando listas enlazadas.  
  
Requisitos:  
- Clase Animal con atributos: Nombre, Edad y Tipo.  
- Lista enlazada que permita agregar animales evitando duplicados.  
- Métodos para mostrar animales de forma iterativa y recursiva.

## Ejemplo de implementación en Python

class Animal:  
 def \_\_init\_\_(self, nombre, edad, tipo):  
 self.nombre = nombre  
 self.edad = edad  
 self.tipo = tipo  
  
 def \_\_str\_\_(self):  
 return f"{self.tipo} - {self.nombre}, {self.edad} años"  
  
class Nodo:  
 def \_\_init\_\_(self, animal):  
 self.animal = animal  
 self.siguiente = None  
  
class ListaAnimales:  
 def \_\_init\_\_(self):  
 self.cabeza = None  
  
 def agregar(self, animal):  
 if self.buscar(animal.nombre, animal.tipo):  
 return  
 nuevo = Nodo(animal)  
 if not self.cabeza:  
 self.cabeza = nuevo  
 else:  
 actual = self.cabeza  
 while actual.siguiente:  
 actual = actual.siguiente  
 actual.siguiente = nuevo  
  
 def buscar(self, nombre, tipo):  
 actual = self.cabeza  
 while actual:  
 if actual.animal.nombre == nombre and actual.animal.tipo == tipo:  
 return True  
 actual = actual.siguiente  
 return False  
  
 def mostrar\_iterativo(self):  
 actual = self.cabeza  
 while actual:  
 print(actual.animal)  
 actual = actual.siguiente  
  
 def mostrar\_recursivo(self, nodo=None):  
 if nodo is None:  
 nodo = self.cabeza  
 if nodo:  
 print(nodo.animal)  
 self.mostrar\_recursivo(nodo.siguiente)

# Ejercicio 2: Sistema de Gestión de Tareas

Objetivo: Diseñar e implementar un sistema básico para gestionar tareas mediante listas enlazadas.  
  
Requisitos:  
- Cada tarea tiene: Descripción, Prioridad y Fecha de vencimiento.  
- Operaciones: agregar, eliminar, mostrar ordenadas, buscar y completar.

## Ejemplo de implementación en Python

from datetime import datetime  
  
class Tarea:  
 def \_\_init\_\_(self, descripcion, prioridad, fecha\_vencimiento):  
 self.descripcion = descripcion  
 self.prioridad = prioridad  
 self.fecha\_vencimiento = datetime.strptime(fecha\_vencimiento, "%Y-%m-%d")  
   
 def \_\_str\_\_(self):  
 return f"[{self.prioridad}] {self.descripcion} (vence: {self.fecha\_vencimiento.date()})"  
  
class NodoTarea:  
 def \_\_init\_\_(self, tarea):  
 self.tarea = tarea  
 self.siguiente = None  
  
class ListaTareas:  
 def \_\_init\_\_(self):  
 self.cabeza = None  
  
 def agregar(self, tarea):  
 nuevo = NodoTarea(tarea)  
 if not self.cabeza:  
 self.cabeza = nuevo  
 else:  
 actual = self.cabeza  
 while actual.siguiente:  
 actual = actual.siguiente  
 actual.siguiente = nuevo  
  
 def eliminar(self, descripcion):  
 actual = self.cabeza  
 previo = None  
 while actual:  
 if actual.tarea.descripcion == descripcion:  
 if previo:  
 previo.siguiente = actual.siguiente  
 else:  
 self.cabeza = actual.siguiente  
 return  
 previo = actual  
 actual = actual.siguiente  
  
 def buscar(self, descripcion):  
 actual = self.cabeza  
 while actual:  
 if actual.tarea.descripcion == descripcion:  
 return actual.tarea  
 actual = actual.siguiente  
 return None  
  
 def mostrar(self):  
 tareas = []  
 actual = self.cabeza  
 while actual:  
 tareas.append(actual.tarea)  
 actual = actual.siguiente  
 tareas.sort(key=lambda t: (t.prioridad, t.fecha\_vencimiento))  
 for tarea in tareas:  
 print(tarea)  
  
 def completar(self, descripcion):  
 self.eliminar(descripcion)

# Pilas

## Definición y características

Una pila es una estructura de datos lineal que sigue el principio LIFO (Last In, First Out): el último elemento en entrar es el primero en salir.  
  
Características principales:  
- Inserción y eliminación en un solo extremo (tope).  
- Acceso restringido (solo al último elemento).  
  
Ventajas:  
- Fácil de implementar.  
- Uso en múltiples algoritmos.  
  
Limitaciones:  
- No permite acceso directo a elementos intermedios.

## Operaciones básicas

- push: insertar un elemento en el tope.  
- pop: eliminar el elemento del tope.  
- peek: ver el elemento del tope sin eliminarlo.  
- isEmpty: verificar si la pila está vacía.

## Pseudocódigo

crear Pila vacía  
  
función push(elemento):  
 agregar elemento al final de la pila  
  
función pop():  
 si la pila está vacía:  
 mostrar "Error: pila vacía"  
 si no:  
 quitar y devolver el último elemento  
  
función peek():  
 si la pila está vacía:  
 mostrar "Error: pila vacía"  
 si no:  
 devolver el último elemento sin quitarlo  
  
función isEmpty():  
 devolver verdadero si la pila no tiene elementos

## Ejemplo en Python

class Pila:  
 def \_\_init\_\_(self):  
 self.elementos = []  
  
 def push(self, dato):  
 self.elementos.append(dato)  
  
 def pop(self):  
 if self.isEmpty():  
 return "Pila vacía"  
 return self.elementos.pop()  
  
 def peek(self):  
 if self.isEmpty():  
 return "Pila vacía"  
 return self.elementos[-1]  
  
 def isEmpty(self):  
 return len(self.elementos) == 0

## Casos de uso comunes

- Deshacer operaciones (Ctrl+Z).  
- Manejo de llamadas a funciones en memoria (Call Stack).  
- Navegación en páginas web (Historial).  
- Algoritmos de Backtracking.  
- Verificación de balanceo de símbolos.

## Ejercicio avanzado: Laberinto con Backtracking

def resolver\_laberinto(laberinto, inicio, salida):  
 pila = [inicio]  
 visitados = set()  
  
 while pila:  
 x, y = pila[-1]  
 if (x, y) == salida:  
 return pila  
 visitados.add((x, y))  
  
 movimientos = [(0,1), (1,0), (0,-1), (-1,0)]  
 encontrado = False  
 for dx, dy in movimientos:  
 nx, ny = x + dx, y + dy  
 if (nx, ny) not in visitados and laberinto[nx][ny] == 0:  
 pila.append((nx, ny))  
 encontrado = True  
 break  
  
 if not encontrado:  
 pila.pop()  
  
 return None

## Ejercicio avanzado: Verificador de balanceo de símbolos

def balanceado(expresion):  
 pila = []  
 pares = {')':'(', '}':'{', ']':'['}  
  
 for i, simbolo in enumerate(expresion):  
 if simbolo in "({[":  
 pila.append(simbolo)  
 elif simbolo in ")}]":  
 if not pila or pila[-1] != pares[simbolo]:  
 return f"Error en posición {i}: '{simbolo}' no balanceado"  
 pila.pop()  
  
 return "Expresión balanceada" if not pila else "Error: faltan cierres"