

## 2. Práctica TAD - Parte 1

TUIA - Programación 2

### Tipo Abstracto de Datos (TAD)

#### Listas

##### Ejercicio 1

Complete la implementación del TAD *Lista* utilizando nodos enlazados. Falta implementar los métodos `index`, `append`, `__len__` y `__str__`. Escriba código de prueba hasta convencerse de que esta clase funciona de manera análoga a las listas de Python.

```
from typing import Any

class _Nodo:
    def __init__(self, dato: Any = None, prox=None):
        self.dato = dato
        self.prox = prox

    def __str__(self):
        return str(self.dato)

def ver_lista(nodo: _Nodo | None) -> None:
    """Recorre todos los nodos a través de sus enlaces, mostrando sus
    contenidos.
    """
    while nodo is not None:
        print(nodo)
        nodo = nodo.prox


class ListaEnlazada:
    """Modela una lista enlazada."""

    def __init__(self) -> None:
        """Crea una lista enlazada vacía."""
        # Referencia al primer nodo (None si la lista está vacía)
        self.prim = None
        # Cantidad de elementos de la lista
        self.len = 0
```

```

def insert(self, i: int, x: Any) -> None:
    """Inserta el elemento x en la posición i.
    Si la posición es inválida, imprime un error y retorna inmediatamente.
    """
    if i < 0 or i > self.len:
        print("Posición inválida")
        return
    nuevo = _Nodo(x)
    if i == 0:
        # Caso particular : insertar al principio
        nuevo.prox = self.prim
        self.prim = nuevo
    else:
        # Buscar el nodo anterior a la posición deseada
        n_ant = self.prim
        for pos in range(1, i):
            n_ant = n_ant.prox
        # Intercalar el nuevo nodo
        nuevo.prox = n_ant.prox
        n_ant.prox = nuevo
    self.len += 1

def pop(self, i: int | None = None) -> Any:
    """Elimina el nodo de la posición i, y devuelve el dato contenido.
    Si i está fuera de rango, se muestra un mensaje de error y se
    retorna inmediatamente. Si no se recibe la posición, devuelve el
    último elemento.
    """
    if i is None:
        i = self.len - 1
    if i < 0 or i >= self.len:
        print(" Posición inválida ")
        return
    if i == 0:
        # Caso particular: saltar la cabecera de la lista
        dato = self.prim.dato
        self.prim = self.prim.prox
    else:
        # Buscar los nodos en las posiciones (i -1) e (i)
        n_ant = self.prim
        n_act = n_ant.prox
        for pos in range(1, i):
            n_ant = n_act
            n_act = n_ant.prox
        # Guardar el dato y descartar el nodo
        dato = n_act.dato
        n_ant.prox = n_act.prox
        self.len -= 1
    return dato

def remove(self, x: Any) -> None:

```

```

"""Borra la primera aparición del valor x en la lista.
Si x no está en la lista, imprime un mensaje de error y retorna
inmediatamente.
"""

if self.len == 0:
    print("La lista esta vacía")
    return
if self.prim.dato == x:
    # Caso particular: saltar la cabecera de la lista
    self.prim = self.prim.prox
else:
    # Buscar el nodo anterior al que contiene a x (n_ant)
    n_ant = self.prim
    n_act = n_ant.prox
    while n_act is not None and n_act.dato != x:
        n_ant = n_act
        n_act = n_ant.prox
    if n_act is None:
        print("El valor no está en la lista.")
        return
    # Descartar el nodo
    n_ant.prox = n_act.prox
    self.len -= 1

def __len__(self):
    """Completar"""
    pass

def __str__(self):
    """Completar"""
    pass

# Faltan index y append

```

### Ejercicio 2

Agregue a `ListaEnlazada` un método `extend` que reciba una `ListaEnlazada` y agregue a la lista actual los elementos que se encuentran en la lista recibida. ¿Puede estimar la complejidad de este método?

### Ejercicio 3

Implemente el método `remover_todos(elemento)` de `ListaEnlazada`, que recibe un elemento y remueve de la lista todas las apariciones del mismo, devolviendo la cantidad de elementos removidos. La lista debe ser recorrida una sola vez.

### Ejercicio 4

Implemente el método `duplicar(elemento)` de `ListaEnlazada`, que recibe un elemento y duplica todas las apariciones del mismo. Ejemplo:

```

L = 1 -> 5 -> 8 -> 8 -> 2 -> 8
L.duplicar(8) = L = 1 -> 5 -> 8 -> 8 -> 8 -> 8 -> 2 -> 8 -> 8

```

### Ejercicio 5

Escriba un método de la clase `ListaEnlazada` que invierta el orden de la lista (es decir, el primer elemento queda como último y viceversa).

### Ejercicio 6

Una desventaja de la implementación del TAD `Lista` que vimos es que es relativamente caro insertar al final de la lista, dado que necesitamos recorrer todos los nodos para poder lograrlo. Esto se puede solucionar utilizando la estructura conocida como `ListaDoblementeEnlazada`. Esta estructura funciona en esencia del mismo modo que la `ListaEnlazada` que ya vimos, pero incorpora un atributo más (`last`) y el siguiente invariante de objeto:

**Invariante:** El atributo `last` es `None` si la lista está vacía. Si no está vacía, `last` apunta al último elemento de la lista.

Dar una implementación del TAD `Lista` utilizando una `ListaDoblementeEnlazada`.

### Ejercicio 7

Implemente el método `filtrar_primos()` en la clase `ListaEnlazada`, que devuelve una nueva lista enlazada con los elementos que sean números primos. La nueva lista debe ser construida recorriendo los nodos una sola vez (es decir, no se puede llamar a `append!`). Ejemplo:

`L = 1 -> 5 -> 8 -> 8 -> 2 -> 8`

`L.filtrar_primos() -> L2 = 5 -> 2`

**Ayuda:** Conviene definir primero una función `es_primo(n)` que reciba un número entero y decida si es primo o no.

### Ejercicio 8

Se tiene una lista enlazada  $L$  de palabras, y se quiere insertar la palabra “mundo” después de todas las apariciones de la palabra “hola”. Defina una función `insertar_palabra_despues(lista, palabra_objetivo, palabra_insertar)` que dada una lista, una palabra objetivo (hola) y una palabra a insertar (mundo) devuelva una nueva lista enlazada donde se agrega la nueva palabra cada vez que se encuentra la palabra objetivo.

Por ejemplo:

`L = “Planificacion” -> “Hola” -> “de” -> “Trece”`

`insertar_palabra_despues(L, “Hola”, “Mundo”) -> L2 = “Planificacion” -> “Hola” -> “Mundo” -> “de” -> “Trece”`

### Ejercicio 9

Se tiene una lista enlazada  $L$  de palabras, y se desea eliminar todas las palabras que contengan una  $\tilde{n}$ . Defina una función `eliminar_palabras_con(lista, carácter)` que dada una lista y un carácter, devuelva una nueva lista enlazada donde se eliminaron las apariciones de palabras conteniendo dicho carácter.

Por ejemplo:

`L = “Ocho” -> “Veinte” -> “Veinticuatro” -> “Hoy”`

`eliminar_palabras_con(L, “V”) -> L2 = “Ocho” -> “Hoy”`