Universidad Americana

Facultad de Ingeniería y Arquitectura

Algoritmos y Estructuras de Datos, Grupo 4

Manejo de Funciones, Clases y Paquetes

Realizado por:

Joaquín Alberto Pérez Zúñiga

Docente:

MSc. César Marín López

1. **__init__.py:**

```python
"""
Implementación de una lista enlazada.
"""

from .node import Node
from .linked_list import LinkedList


__all__ = [
    "Node",
    "LinkedList"
]
```

2. **node.py:**

```python
"""
Implementación de un nodo en una lista enlazada.
"""

from typing import Any, Optional


class Node:
    """
    Representa un nodo en una lista enlazada,
    con el dato que almacena, y el proximo elemento en la lista.
    """

    def __init__(self, data: Any, next_node: Optional["Node"] = None):
        self.data = data
        self.next = next_node

```

3. linked_list.py:

```python
"""
Implementación personalizada de una lista enlazada.
"""

from typing import Any, Optional

from .node import Node


class LinkedList:
    """
    Implementación personalizada de una lista enlazada.
    """

    def __init__(self):
        self.head: Optional[Node] = None

    def __str__(self) -> str:
        current = self.head

        list_str = "[ "
        while current is not None:
            list_str += f"{current.data}{"," if current.next is not None else ""}"
        list_str += " ]"

        return list_str

    def is_empty(self) -> bool:
        """
        Determina si la lista esta vacía.
        """

        return self.head is None

    def add_at_front(self, data: Any) -> None:
        """
        Agrega un nuevo elemento al inicio de la lista enlazada.
        """

        self.head = Node(data=data, next_node=self.head)
```

```python

def add_at_end(self, data: Any) -> None:
    """
    Agrega un nuevo elemento al final de la lista enlazada.
    """

    if self.is_empty():
        self.head = Node(data)
        return

    current = self.head
    while True:
        if current.next is not None:  # type: ignore
            current = current.next  # type: ignore
        else:
            current.next = Node(data)  # type: ignore
            return

def search(self, data: Any) -> Optional[Node]:
    """
    Busca el valor de un nodo en la lista.
    """

    elem_actual = self.head

    while True:
        if elem_actual.data == data:  # type: ignore
            return elem_actual
        if elem_actual.next is None:  # type: ignore
            return None

        elem_actual = elem_actual.next  # type: ignore

def delete(self, data: Any) -> bool:
    """
    Elimina el valor de un nodo.
    """

    current = self.head
    previous = None

    while current is not None and current.data != data:
        previous = current
        current = current.next
        if previous is None:
            self.head = current.next
        elif current is not None:
            previous.next = current.next
            current.next = None

    return True

```