

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

Nicolás Rafael Laguna Vallejos

Docente:

MSc. César Marín López

1. `__init__.py`:

```
1  """
2  Implementación de una lista enlazada.
3  """
4
5  from .node import Node
6  from .linked_list import LinkedList
7
8
9  __all__ = [
10     "Node",
11     "LinkedList"
12 ]
```

2. `node.py`:

```
1  """
2  Implementación de un nodo en una lista enlazada.
3  """
4
5  from typing import Any, Optional
6
7
8  class Node:
9      """
10     Representa un nodo en una lista enlazada,
11     con el dato que almacena, y el proximo elemento en la lista.
12     """
13
14     def __init__(self, data: Any, next_node: Optional["Node"] = None):
15         self.data = data
16         self.next = next_node
17
```

3. linked_list.py:

```
1  """
2  Implementación personalizada de una lista enlazada.
3  """
4
5  from typing import Any, Optional
6
7  from .node import Node
8
9
10 class LinkedList:
11     """
12     Implementación personalizada de una lista enlazada.
13     """
14
15     def __init__(self):
16         self.head: Optional[Node] = None
17
18     def __str__(self) -> str:
19         current = self.head
20
21         list_str = "[ "
22         while current is not None:
23             list_str += f"{current.data}{"", " if current.next is not None else ""}"
24             list_str += " ]"
25             current = current.next
26
27         return list_str
28
29     def is_empty(self) -> bool:
30         """
31         Determina si la lista esta vacía.
32         """
33
34         return self.head is None
35
36     def add_at_front(self, data: Any) -> None:
37         """
38         Agrega un nuevo elemento al inicio de la lista enlazada.
39         """
40
41         self.head = Node(data=data, next_node=self.head)
```



```
1
2 def add_at_end(self, data: Any) -> None:
3     """
4     Agrega un nuevo elemento al final de la lista enlazada.
5     """
6
7     if self.is_empty():
8         self.head = Node(data)
9         return
10
11     current = self.head
12     while True:
13         if current.next is not None: # type: ignore
14             current = current.next # type: ignore
15         else:
16             current.next = Node(data) # type: ignore
17             return
18
19 def search(self, data: Any) -> Optional[Node]:
20     """
21     Busca el valor de un nodo en la lista.
22     """
23
24     elem_actual = self.head
25
26     while True:
27         if elem_actual.data == data: # type: ignore
28             return elem_actual
29         if elem_actual.next is None: # type: ignore
30             return None
31
32         elem_actual = elem_actual.next # type: ignore
33
34 def delete(self, data: Any) -> bool:
35     """
36     Elimina el valor de un nodo.
37     """
38
39     current = self.head
40     previous = None
41
42     while current is not None and current.data != data:
43         previous = current
44         current = current.next
45         if previous is None:
46             self.head = current.next
47         elif current is not None:
48             previous.next = current.next
49             current.next = None
50
51     return True
52
```