DOCUMENTO TÉCNICO

Integrantes:

* Diego Cárdenas
* Felipe Calvache
* Zayra Gutiérrez

Especificación:

Se pide implementar la función Delete en la LinkedList

Entrada:

Se ingresa una lista y un valor que se quiere eliminar de la lista dada (los elementos de la lista son nodos).

Salida**:**

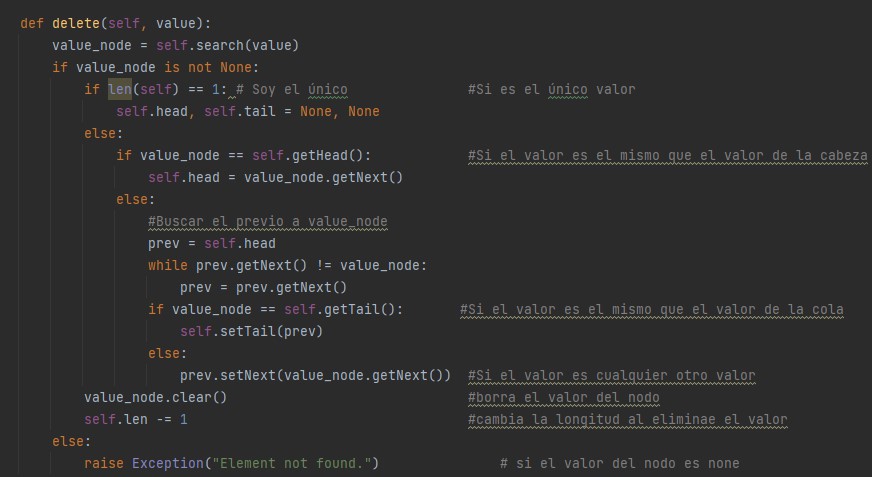
La salida corresponderá a la lista original sin el valor dado para eliminar.

Diseño**:**

Estrategia

La táctica empleada para llevar a cabo la acción de "eliminar" consiste en abordarla mediante diferentes casos: Si el valor que se desea eliminar es el único elemento presente en la lista, entonces la lista quedará vacía. Si el valor que se desea eliminar es la cabeza de la lista, se asignará como nueva cabeza el valor siguiente al valor original. Si el valor a eliminar es el último elemento de la lista, la nueva cola será el valor previo al elemento original. En caso de que el valor a eliminar sea cualquier otro valor distinto a los mencionados anteriormente, se buscará el valor previo y se enlazará con su nuevo valor siguiente, que será el siguiente al valor a eliminar, y finalmente se eliminará el valor deseado.

Algoritmo y Documentación



# Casos de Prueba

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| [1, 2, 3, 4 ,5]  4 | Caso típico | [1, 2, 3, 5] |

Punto\_2

Especificación:

Realizar la implementación de Pila y Cola (Queue, Stack) en donde en su definición estructural se reuse la definición realizada en el punto 1. En las operaciones CRUD recuerde mantener la pólitica de cada estructura (LIFO - FIFO)

Entrada:

Se ingresa una lista donde todos sus elementos son de tipo nodo

Salida**:**

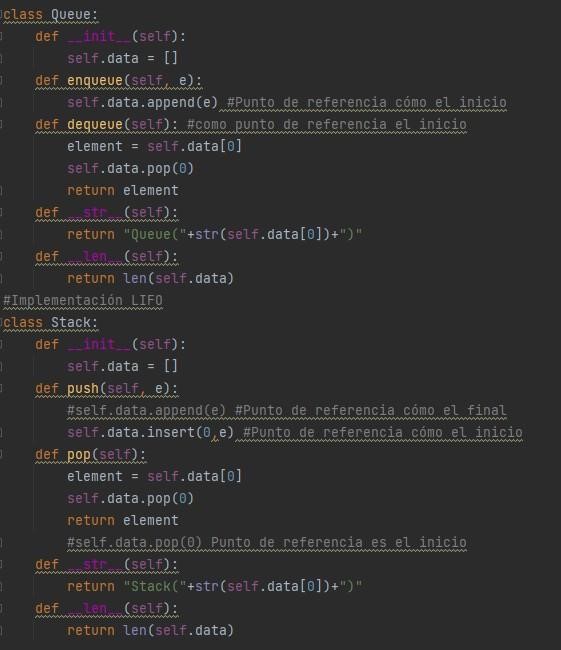
Se retorna una lista doblemente enlazada con las funciones CRUD

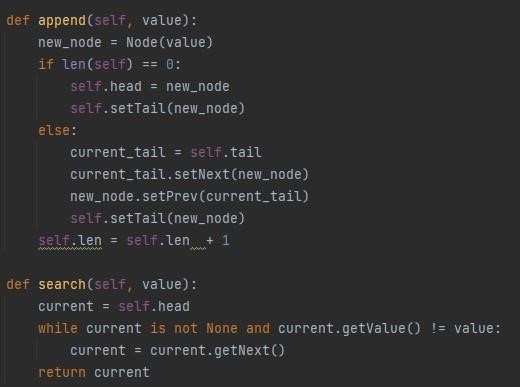
Diseño**:**

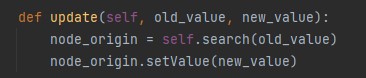
Estrategia

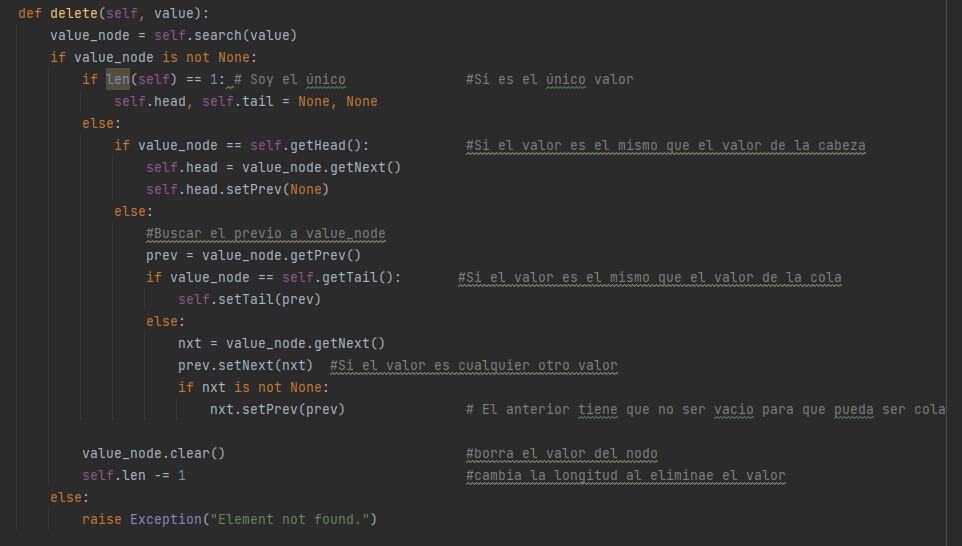
La estrategia empleada para implementar este código comienza utilizando la estructura de LinkedList ya existente. Se crea una nueva referencia denominada "Tail" y se le asigna la conexión con el nodo previo, utilizando esta nueva referencia. actualizamos las funciones CRUD implementando Tail como punto de comparación para el nodo anterior, permitiendo de esta manera tener una DoubleLinkedList

Algoritmo y Documentación









# Casos de Prueba

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| list = DoubleLinkedList(3, 4, 5, 6]) list.delete(4) | Comprobar el funcionamiento de la función delete con el doble enlazamiento | [(3) -->(5) -->(6) -->None] |
| list = DoubleLinkedList([3, 4, 5, 6]) list.update(4, 2) | Comprobar la actualización de los valores de los nodos con el doble enlazamiento | [(3) -->(2) -->(5) -->(6) --  >None] |

Punto\_3

Especificación:

Implementar la operación de LinkedList Reverse, definida cómo una función que invierte los enlaces de los nodos de la LinkedList.

Entrada:

IN: [1,2,3,4,5]

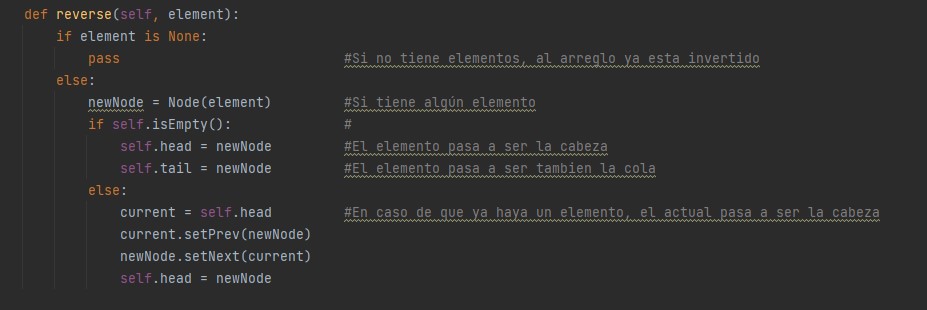
Salida**:** OUT: [5,4,3,2,1]

Diseño**:**

Estrategia

Para implementar la función "reverse", utilizamos la definición estructural desarrollada previamente. La estrategia consiste en abordarla por casos. Primero, se verifica que el arreglo no esté vacío. Si la lista que se está generando tiene solo un elemento, este se convierte en la cabeza y la cola. En el caso de que ya existan más de un elemento, la cabeza se convierte en el nuevo elemento y la cola se establece como el elemento que estaba en la lista enlazada previamente.

Algoritmo y Documentación



# Casos de Prueba

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Justificación** | **Salida** |
| [1] | Lista unitaria | [1] |
| [1,2,3,4,5] | Lista que pide el ejercicio | [5,4,3,2,1] |