***Python Classes and Linked Lists***

La lista en Python es una linked list.

Una array es almacenador de elementos del mismo tipo con una capacidad fija.

Gráfico

Descripción generada automáticamente

***Diferencia entre array y listas:***

1. Los arrays solo almacenan datos del mismo tipo, las listas almacenan de diferentes tipos.
2. Los arrays tiene capacidades estáticas, mientras que las listas son dinámicas.
3. Las listas permiten hacer muchas operaciones como, (insertar, eliminar, modificar) mientras que los arrays no permiten (insertar, eliminar).
4. Las arrays guardan sus elementos en secuencia, uno al lado de otro. Las listas no están sujetas a esta regla, porque se puede insertar o eliminar.





El NODO A contiene su información y la del NODO B, y así sucesivamente, pero el NODO A no tiene ni idea sobre el NODO C.

La linked list siempre sabe cual es el primer NODO, en este caso el NODO A y además sabe su LONGITUD.

Si queremos buscar el NODO C necesitamos recorrerla de forma lineal, empezando por el NODO A hasta llegar al NODO C.

La forma de tu ELIMINAR en un NODO en una linked list de un solo sentido es quitando el CONECTOR. Ejemplo:

1. Queremos eliminar el NODO 14:

Imagen que contiene Gráfico

Descripción generada automáticamente

1. Buscamos el NODO anterior al NODO 14, en este caso el NODO 15:

Imagen que contiene Gráfico

Descripción generada automáticamente

1. Y le cortamos el CONECTOR con el NODO 14.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

1. Pero como no podemos dejarlo un NODO sin un CONECTOR, habiendo mas NODOS, pues lo CONECTAMOS al NODO SIGUIENTE, en este caso, EL NODO 12.

Diagrama

Descripción generada automáticamente

De tal forma que el recorrido seria de la siguiente manera: NODO 15 >>>> NODO 12 >>>> NODO 10 >>>> NULL, dejando al NODO 14 fuera.

La forma de AGREGAR es al final de la linked list:

1. Buscamos el ultimo NODO, en este caso el NODO C.

Diagrama

Descripción generada automáticamente

1. Y le agregamos el NUEVO NODO justo delante.

Imagen que contiene objeto, reloj

Descripción generada automáticamente

***Programación:***

Lo primero es programar el NODO y esto se hace con una CLASS.

# ESTO ES PARA CREAR EL NODO EN SI, CON LA INFORMACION COMO VALUE

class Node:

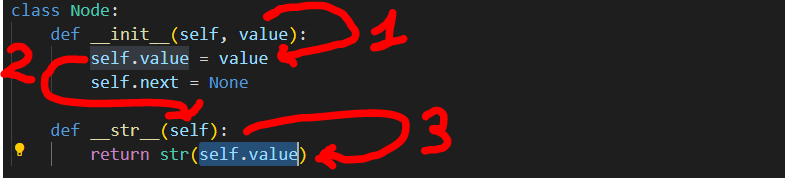
    def \_\_init\_\_(self, value):

        self.value = value

        self.next = None

    def \_\_str\_\_(self): # esto es para devolver el Nodo o el Value en formato STR

        return str(self.value) # aquí está capturando el self.value y volviéndolo un STR



Creación de la LINKED LIST:

First = es el primer elemento o el Head.

Size = es la longitud de la lista.

Los dos valores que las LINKED LIST siempre saben.

class linkedList:

    def \_\_init\_\_(self):

        self.First = None

        self.Size = 0

Nota: COMO FUNCIONA EL SELF: Self es un conector de la variable, en este caso, self es el enlace entre Mylist y la función Append



SELF == MyList // VALUE == 1



Creación de la Funcionalidad de la LINKED LIST:

#CREACION DEL NODO, con la informacion (DATA) y el nodo siguiente (NEXT)

class Node:

    def \_\_init\_\_ (self, data = None, next = None):

        self.data = data

        self.next = next

#CREACION DEL HEAD DE LA LINKEDLIST

class LinkedList:

    def \_\_init\_\_(self):

        self.head = None

    def insert\_at\_begining (self, data):

        node = Node (data, self.head)

        self.head = node

    ll = LinkedList ()

    ll.insert\_at\_begining(5)

    ll.insert\_at\_begining(89)

GUIA DE EJECUCION:

Primero se ejecuta la clase.

Después llamamos a la función INSERT\_AT\_BEGINING(5)

Luego el numero 5 pasa a la variable data dentro de la DEF

Se ejecuta la variable NODE, la cual creara un NODE(DATA, SELF. HEAD) = NODE(5, SELF.HEAD)

El 5 pasara la variable DATA y el SELF.HEAD a la variable NEXT = NODE(5, NONE)

Una vez fue creado el NODO, se actualiza el valor de SELF.HEAD a 5 (NODE(5, SELF.HEAD))

Esto hace que SELF.HEAD sea el primer nodo con una data de 5.

Se llama a la función INSERT\_AT\_BEGINING(89)

Luego el numero 89 pasa a la variable data dentro de la DEF

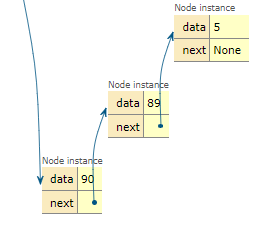
Se ejecuta la variable NODE, la cual creara un NODE(DATA, SELF. HEAD) = NODE(89, 5)

Recordemos que el SELF.HEAD, actualizo su valor a 5.

El 89 pasara la variable DATA y el SELF.HEAD a la variable NEXT = NODE(89, 5)

Una vez fue creado el NODO, se actualiza el valor de SELF.HEAD a 89 (NODE(89, 5))

Esto hace que SELF.HEAD sea el primer nodo con una data de 89, y para acceder al primer elemento que ahora es el 2DO, es: HEAD.NEXT.

 EL head, siempre pasara a ser el next del nodo nuevo.

    def insert\_at\_end(self, data):

        # si el Next del Nodo Raiz de turno es None

        if self.head is None:

            #se actualiza el head al nuevo nodo.

            self.head = Node(data, None)

            return

        itr = self.head

        #esto es para iterar entre todos los next del head

        while itr.next:

            itr = itr.next

        # cuando no haya mas, el while saltara False y se creara el siguiente en el Next del

        itr.next = Node(data, None)

Notas: el primer if, no se va a ejecutar porque el ultimo head es el 89, de ahí que el ciclo While, itere entre todos los next, ¿porque todos los next? Porque todos los head son un nodo con el valor next dentro de él.

Diagrama

Descripción generada automáticamente

    def insert\_values(self, data\_list):

        # esto es para establecer que no hay una cabezaa

        self.head = None

        for data in data\_list:

            self.insert\_at\_end(data)

SI LO EJECUTAMOS ASI, SE BORRARÁN TODOS LOS VALORES ANTERIORES, PORQUE RECORDEMOS QUE LOS ELEMENTOS (5,89, 90) ESTAN EN LA VARIABLE, SELF.HEAD, Y EN LA FUNCION INSERT.VALUES, SELF.HEAD VALE NONE.

if \_\_name\_\_ == '\_\_main\_\_':

    ll = LinkedList ()

    ll.insert\_at\_begining(5)

    ll.insert\_at\_begining(89)

    ll.insert\_at\_end(90)

    ll.insert\_values(["roddy", "jenny"])

    ll.print()



ANTES DEL SELF.HEAD = None

Diagrama

Descripción generada automáticamente

DESPUES DEL SELF.HEAD = None

Diagrama

Descripción generada automáticamente

COMO TERMINA:

Diagrama

Descripción generada automáticamente

Primero se insertan los valores de las listas y luego se ejecuta los demás.

if \_\_name\_\_ == '\_\_main\_\_':

    ll = LinkedList ()

    ll.insert\_values(["roddy", "jenny"])

    ll.insert\_at\_begining(5)

    ll.insert\_at\_begining(89)

    ll.insert\_at\_end(90)

    ll.print()



Diagrama

Descripción generada automáticamente

#EN ESTE CASO, EL WHILE ESTA ITERANDO ENTRE TODOS LOS ELEMENTOS ANTERIORMENTE CREADOS, Y QUE ESTAN EN HEAD.

#LUEGO CAMBIA SU VALOR AL DEL SIGUIENTE Y ESTO SE CUMPLIRA HASTA QUE NO HAYA ELEMENTOS, EN EL NEXT.

    def get\_length(self):

        count = 0

        itr = self.head

        while itr:

            count += 1

            itr = itr.next

        return count

    ll = LinkedList ()

    ll.insert\_values(["roddy", "jenny"])

    ll.print()

    print(ll.get\_length())



# si el número es mayor de 1, se ejecutara el código del final, el cual lo que hace es mover el head al siguiente elemento.

        itr = self.head >>>> itr = itr.next

    def remove\_at(self, index):

        if index < 0 or index >= self.get\_length():

            raise Exception("invalid index")

        if index == 0:

            self.head = self.head.next

            return

        count = 0

        itr = self.head

        while itr:

            if count == index - 1:

                itr.next = itr.next.next

                break

            itr = itr.next

            count += 1

AQUI HAY UNA ACTUALIZACION DE VALORES USANDO EL itr.next

    def insert\_at(self, index, data):

        if index < 0 or index > self.get\_length():

            raise Exception("Invalid Index")

        if index == 0:

            self.insert\_at\_begining(data)

            return

        count = 0

        itr = self.head

        while itr:

            if count == index - 1:

                node = Node(data, itr.next)

                itr.next = node

                break

            itr = itr.next

            count += 1

***Entonces, una vez termine cada llamada de Insert\_at\_begining, ese valor resultante se almacenará en la Class.***

#COMO IMPRIMIR LA SECUENCIA

    def print(self):

        if self.head is None:

            print("My LinkedList is Empty")

            return

        itr = self.head

        llistr = ''

        while itr:

            llistr += str(itr.data) + " --> "

            itr = itr.next

        print(llistr)

***EJECICIO:***

In LinkedList class that we implemented in our tutorial add following two methods.

def insert\_after\_value(self, data\_after, data\_to\_insert):

# Search for first occurance of data\_after value in linked list

# Now insert data\_to\_insert after data\_after node

    def insert\_after\_value (self, data\_after, data\_to\_insert):

        itr = self.head

        if itr.data == data\_after:

            node = Node (data\_to\_insert, itr.next)

            itr.next = node

            return

        while itr:

            if itr.next.data == data\_after:

                node = Node (data\_to\_insert, itr.next.next)

                itr.next.next = node

                break

            itr = itr.next

    ll = LinkedList()

    ll.insert\_values(["banana","mango","grapes","orange"])

    ll.print()

    ll.insert\_after\_value("banana","apple") # insert apple after mango

    ll.print()



***FORMA CORRECTA DE HACER:***

    def insert\_after\_value(self, data\_after, data\_to\_insert):

        if self.head is None:

            return

        if self.head.data==data\_after:

            self.head.next = Node(data\_to\_insert,self.head.next)

            return

        itr = self.head

        while itr:

            if itr.data == data\_after:

                itr.next = Node(data\_to\_insert, itr.next)

                break

            itr = itr.next

def remove\_by\_value(self, data):

# Remove first node that contains data

    def remove\_by\_value(self, data):

        try:

            itr = self.head

            if itr.data == data:

                    self.head = itr.next

                    return

            while itr:

                if itr.next.data == data:

                    # Orange = Grapes

                    itr.next =  itr.next.next

                    break

                itr = itr.next

        except AttributeError:

            print(f"el objeto {data} no esta en la lista")

***FORMA CORRECTA DE HACER:***

    def remove\_by\_value(self, data):

        if self.head is None:

            return

        if self.head.data == data:

            self.head = self.head.next

            return

        itr = self.head

        while itr.next:

            if itr.next.data == data:

                itr.next = itr.next.next

                break

            itr = itr.next

ll.remove\_by\_value("banana") # remove orange from linked list

    ll.remove\_by\_value("mango")

    ll.remove\_by\_value("apple")

    ll.remove\_by\_value("grapes")

    ll.print()



    ll.remove\_by\_value("figs")

    ll.remove\_by\_value("figs")



***DOUBLE LINKED LIST***