In [2]:

```python
# 1. Cree un nuevo archivo Python y vuelva a crear las clases Node y SList
class SLNode:
    def __init__(self, val):
        self.value = val
        self.next = None

#1. Cree un nuevo archivo Python y vuelva a crear las clases Node y SList
class SList:
    def __init__(self):
        self.head = None

#2.Agregue el método add_to_front a su clase SList
    def add_to_front(self, val):                # agrega la linea, toma un valor
        new_node = SLNode(val)              # crea una instancia de la clase Node usando el valor dado
        current_head = self.head              # salva la cabecera actual en una variable
        new_node.next = current_head    # Coloca el proximo nodo en la lista de la cabecera actual
        self.head = new_node                  # Coloca la lista de la cabecera al nodo que se creó en el pa
        return self                                       # return self para permitir las cadenas

#3. Agregue el método print_values ••a su clase SList
    def print_values(self):
        runner = self.head            # un puntero al primer nodo de la lista
        while (runner != None):    # iterando mientras el corredor es un nodo y no ninguno
            print(runner.value)        # imprimir el valor del nodo actual
            runner = runner.next    # Establecer el corredor a su vecino
        return self                           # Una vez que el bucle está terminado, regrese a sí mismo para

#4. Agregue el método add_to_back a su clase SList
    def add_to_back(self, val):     # acepta un valor
        if self.head == None:        # si la lista está vacia
            self.add_to_front(val)     # ejecuta el método add_to_front
            return self                      # asegurémonos de que el resto de esta función no suceda si ag

        new_node = SLNode(val)  # crea una nueva instancia de nuestra clase Node con el valor dado
        runner = self.head    # establece un iterador para que comience al principio de la lista
        while (runner.next != None):  # iterador hasta que el iterador no tenga un vecino
            runner = runner.next # Incrementa el corredor al siguiente nodo de la lista.
        runner.next = new_node  # Incrementa el corredor al siguiente nodo de la lista.
        return self


#5. Agregue el método remove_from_front a su clase SList
```

```
    def remove_from_front(self, val):        # elimina el primer nodo y devuelve su  valor
        new_node = SLNode(val)               # crea una instancia de la clase Node usando el valor dado
        current_head = self.head               # salva la cabecera actual en una variable
        new_node.next = current_head    # Coloca el proximo nodo en la lista de la cabecera actual
        self.head = new_node                   # Coloca la lista de la cabecera al nodo que se creó en el pa
        return self

my_list = SList()   # crear una nueva instancia de una lista
my_list.add_to_front("son").add_to_front("Listas enlazadas").add_to_back("divertidas!").print_values()    #
```

```
Listas enlazadas
son
divertidas!
```

Out[2]: <__main__.SList at 0x24f7ec52370>

In [1]:

```python
# 1. Cree un nuevo archivo Python y vuelva a crear las clases Node y SList
class SLNode:
    def __init__(self, val):
        self.value = val
        self.next = None

#1. Cree un nuevo archivo Python y vuelva a crear las clases Node y SList
class SList:
    def __init__(self):
        self.head = None

#2.Agregue el método add_to_front a su clase SList
    def add_to_front(self, val):                # agrega la linea, toma un valor
        new_node = SLNode(val)              # crea una instancia de la clase Node usando el valor dado
        current_head = self.head              # salva la cabecera actual en una variable
        new_node.next = current_head    # Coloca el proximo nodo en la lista de la cabecera actual
        self.head = new_node                  # Coloca la lista de la cabecera al nodo que se creó en el pa
        return self                                       # return self para permitir las cadenas

#3. Agregue el método print_values ••a su clase SList
    def print_values(self):
        runner = self.head            # un puntero al primer nodo de la lista
        while (runner != None):    # iterando mientras el corredor es un nodo y no ninguno
            print(runner.value)        # imprimir el valor del nodo actual
            runner = runner.next    # Establecer el corredor a su vecino
        return self                           # Una vez que el bucle está terminado, regrese a sí mismo para

#4. Agregue el método add_to_back a su clase SList
    def add_to_back(self, val):     # acepta un valor
        if self.head == None:        # si la lista está vacia
            self.add_to_front(val)     # ejecuta el método add_to_front
            return self                      # asegurémonos de que el resto de esta función no suceda si ag

        new_node = SLNode(val)  # crea una nueva instancia de nuestra clase Node con el valor dado
        runner = self.head    # establece un iterador para que comience al principio de la lista
        while (runner.next != None):  # iterador hasta que el iterador no tenga un vecino
            runner = runner.next # Incrementa el corredor al siguiente nodo de la lista.
        runner.next = new_node  # Incrementa el corredor al siguiente nodo de la lista.
        return self

#NINJA BONUS: complete el método remove_from_front
    def remove_from_front(self):
```

```
        runner = self.head
        self.head = runner.next
        return self

#NINJA BONUS: complete el método remove_from_back
    def remove_from_back(self):
        if(self.head!=None and self.head.next!=None):
            runner = self.head
            while(runner.next.next != None):
                runner = runner.next
            runner.next=None
        elif(self.head.next==None):
            self.head.value=None
        return self

#NINJA BONUS: complete el método remove_val
    def remove_val(self, val):
        runner = self.head
        #Primer nodo eliminado
        if runner.value == val:
            self.head = runner.next
            return self
        #Eliminar el nodo con el valor en medio de la lista
        while (runner.next.value != val):
            runner = runner.next
            temp = runner.next
        runner.next = temp.next
        return self

#SENSEI BONUS: complete el método insert_at
    def insert_at(self, val, n):
        runner = self.head
        if n==0:
            self.add_to_front(val)
        else:
            contador = 1
            while contador<n:
                runner = runner.next
                contador += 1
            temp = runner.next
            nuevo = SLNode(val)
            runner.next = nuevo
            nuevo.next = temp
```

```
        return self



my_list = SList()   # crear una nueva instancia de una lista
my_list.add_to_back("1").add_to_back("2").add_to_back("3").add_to_back("4").insert_at("5", 4).print_values(

#Practique lo anterior en código y en papel / pizarra. ¡Entonces intente escribir estos métodos desde cero
#Practique lo anterior en su computadora y en papel o en una pizarra. ¡Entonces intente escribir estos méto
#SENSEI BONUS: considere y tenga en cuenta los casos límite para todos los métodos anteriores.
```

```
1
2
3
4
5
```

Out[1]: <__main__.SList at 0x2716415be20>

In [ ]:

```
1
```