

# SIMULACIÓN E IMPLEMENTACIÓN DE OPERACIONES PRINCIPALES SOBRE LISTAS ENLAZADAS

*Por:*  
*Joaquín Pérez*  
*Nicolas Laguna*  
*Solieth Trejos*

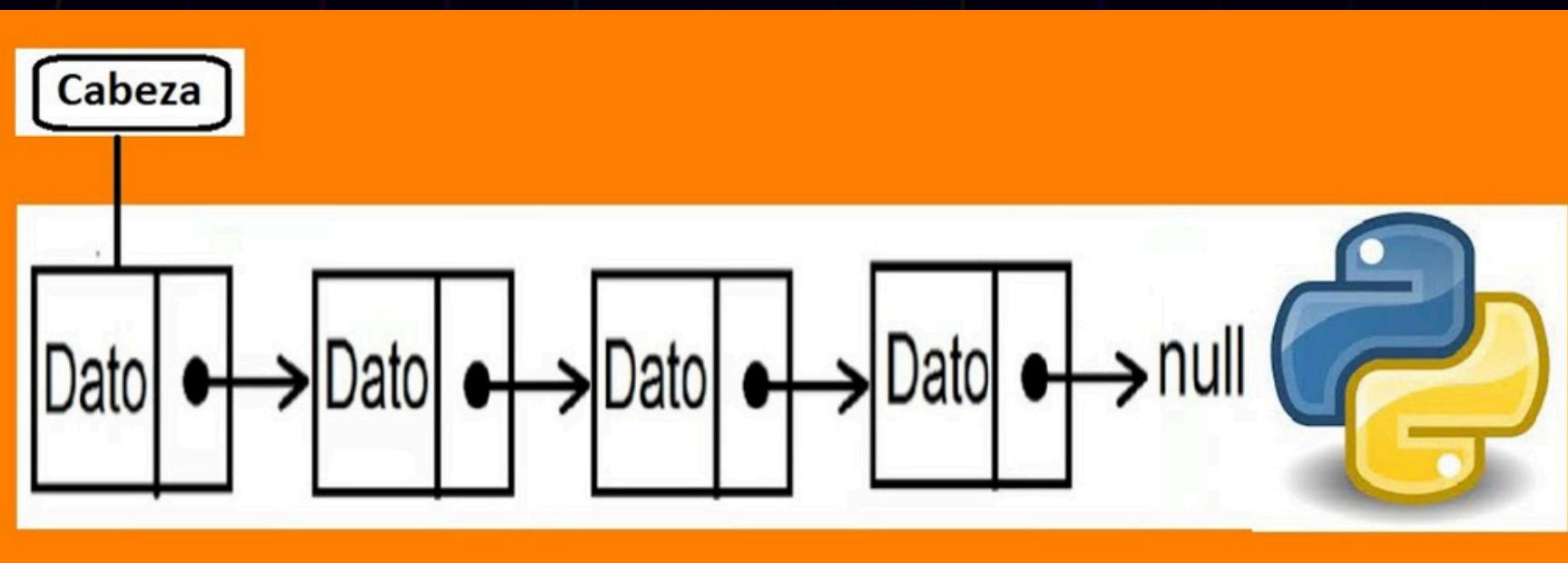


**¿QUÉ ES UNA  
LISTA  
ENLAZADA?**

Una lista enlazada es una estructura de datos lineal donde cada elemento (llamado nodo) contiene dos partes:

1. Los datos a almacenar
2. Una referencia (enlace) al siguiente nodo en la secuencia

A diferencia de los arrays/listas normales de Python, en una lista enlazada los elementos no están almacenados en posiciones contiguas de memoria.



## 1. GESTIÓN DE MEMORIA

Las listas enlazadas pueden ser utilizadas para gestionar la memoria de manera eficiente. En lugar de utilizar una lista indexada que requiere una cantidad fija de memoria, las listas enlazadas pueden crecer o disminuir dinámicamente.

## 2. IMPLEMENTACIÓN DE ESTRUCTURAS DE DATOS

Las listas enlazadas pueden ser utilizadas para implementar otras estructuras de datos como:

- Pilas (stacks): estructura de datos que sigue el orden de llegada, es decir, el último elemento que se agrega es el primero en salir.
- Colas (queues): estructura de datos que sigue el orden de llegada, es decir, el primer elemento que se agrega es el primero en salir.

## 3. MANIPULACIÓN DE DATOS

Las listas enlazadas pueden ser utilizadas para manipular datos de manera eficiente, especialmente cuando se necesita insertar o eliminar elementos en posiciones específicas.

# APLICACIONES DE LISTAS ENLAZADAS

## 4. ÁLGEBRA LINEAL

Las listas enlazadas pueden ser utilizadas para representar matrices dispersas, que son matrices que contienen la mayor parte de los elementos iguales a cero.

## 5. GRÁFICOS Y ÁRBOLES

Las listas enlazadas pueden ser utilizadas para representar gráficos y árboles, que son estructuras de datos que se utilizan para modelar relaciones entre objetos.

## 6. SISTEMA DE GESTIÓN DE ARCHIVOS

Las listas enlazadas pueden ser utilizadas en sistemas de gestión de archivos para mantener un registro de los archivos y su ubicación en el disco.



# OPERACIONES BÁSICAS

# INSERCIÓN

AL INICIO:

```
● ○ ●  
1 def add_at_front(self, data: T) -> None:  
2     """  
3     Agrega un nuevo elemento al inicio de la lista enlazada.  
4     """  
5  
6     self.head = Node(data=data, next_node=self.head)  
7     self.size += 1
```

AL FINAL:

```
● ○ ●  
1 def add_at_end(self, data: T) -> None:  
2     """  
3     Agrega un nuevo elemento al final de la lista enlazada.  
4     """  
5  
6     new_node = Node(data)  
7  
8     if self.is_empty():  
9         self.head = new_node  
10        self.size += 1  
11        return  
12  
13     current = self.head  
14     while True:  
15         if current.next is not None:  
16             current = current.next  
17         else:  
18             current.next = new_node  
19             self.size += 1  
20             return
```

# ELIMINACIÓN

AL INICIO:

```
● ● ●  
1 def delete_first(self) -> None:  
2     """  
3     Elimina el primer nodo de la lista.  
4     """  
5  
6     if self.is_empty():  
7         return  
8  
9     if self.head.next is None:  
10        self.head = None  
11    else:  
12        self.head = self.head.next
```

AL FINAL:

```
● ● ●  
1 def delete_last(self) -> None:  
2     """  
3     Elimina el último nodo de la lista.  
4     """  
5  
6     if self.is_empty():  
7         return  
8     if self.size == 1:  
9         self.delete_first()  
10        return  
11  
12     previous = None  
13     current = self.head  
14  
15     while True:  
16         if current.next is None:  
17             previous.next = None  
18             return  
19         previous = current  
20         current = current.next
```

# RECORRIDO



```
1 def __str__(self) -> str:
2     self_str = "[ "
3
4     current = self.head
5     while current is not None:
6         if current.next is not None:
7             self_str += f"{str(current.data)}, "
8         else:
9             self_str += f"{str(current.data)} ]"
10        current = current.next
11
12    return self_str
```

# BÚSQUEDA



```
1 def search(self, data: T) -> Optional[Node]:  
2     """  
3     Busca el valor de un nodo en la lista.  
4     """  
5  
5     if self.is_empty():  
6         return None  
7  
8     current = self.head  
9     while current is not None:  
10        if current.data == data:  
11            return current  
12        current = current.next  
13  
14    return None
```

# ACTUALIZACIÓN

```
● ● ●  
1 def update_node(self, index: int, new_data: T) -> None:  
2     """  
3         Actualiza el valor de un nodo en la lista.  
4     """  
5  
6     if self.is_empty():  
7         raise IndexError("Índice inválido: no hay elementos en la lista.")  
8     if index < 0:  
9         raise IndexError("Índice inválido: debe ser mayor o igual a cero.")  
10    if index >= self.size:  
11        raise IndexError(  
12            "Índice inválido: es fuera de límites; no corresponde a un nodo."  
13        )  
14  
15    i = 0  
16    current = self.head  
17    while current is not None and i < index:  
18        i += 1  
19        current = current.next  
20  
21    if current is not None:  
22        current.data = new_data
```

# PROBLEMA #2



1     ""

2 Una escuela de educación primaria requiere un algoritmo que muestre los datos de los  
3 estudiantes de un salón de clase ordenados de forma ascendente, según un parámetro  
4 indicado; este parámetro puede ser cualquiera de los siguientes campos:  
5 carnet, nombres, apellidos, genero, peso, estatura, promedio.

6     ""

# PROBLEMA #2



- 1     """
- 2     Se requiere automatizar un mapa que contiene las estaciones de una ruta previamente establecida para una aplicación que indique, a partir de un punto de la ruta, el tiempo estimado para llegar a un destino determinado de la misma.
- 3     """
- 4
- 5
- 6

# PROBLEMA #3



1   """

2 Una clínica recibe pacientes en orden de llegada. Cada paciente debe ser ingresado al  
3 sistema con los siguientes datos: nombre completo, edad, síntoma principal y prioridad  
4 (de 1 a 5). El sistema debe permitir insertar nuevos pacientes, recorrer la lista  
5 para mostrar el orden de atención, y eliminar a un paciente una vez atendido.

6   """

# PROBLEMA #4



1     ""

2   Se desea implementar el historial de acciones realizadas por un usuario en un  
3   editor de texto (como escribir, borrar, pegar, copiar). Cada acción debe  
4   guardarse en orden y poder recorrerlas en ambas direcciones,  
5   simulando las acciones de Deshacer y Rehacer.

6     ""



**iGRACIAS!**