República de Panamá

Universidad Interamericana de Panamá

Faculta de Ingeniería

**TAREA INICIAL**

# Estructura de Datos II


Estudiante:

**Keisy Morales 4-807-2495**

Profesor:

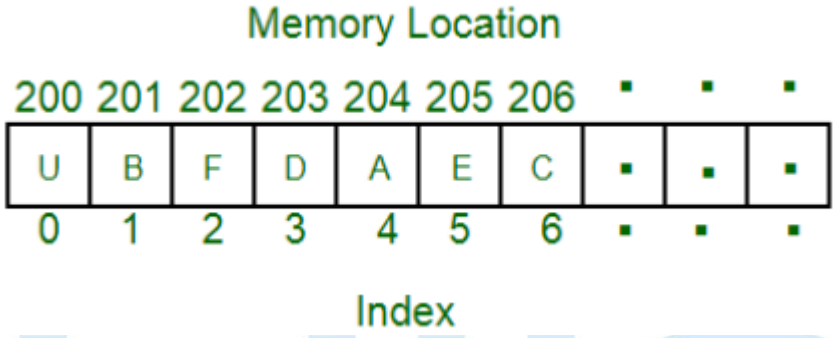
**Leonardo Esqueda**


Fecha:

04/06/2022

**Diferencias entre Arreglos y las Matrices**

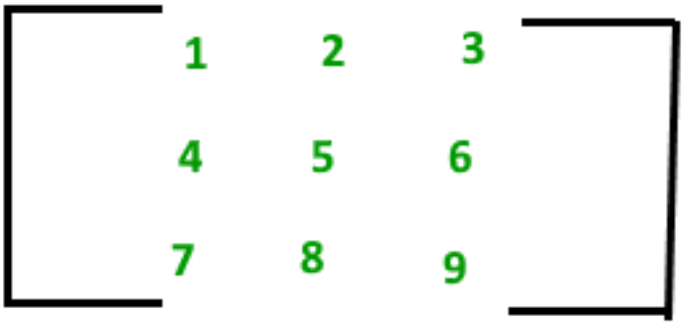
- **Un arreglo** es una colección de elementos almacenados en ubicaciones de memoria contiguas. La idea es almacenar varios artículos del mismo tipo juntos. Esto facilita el cálculo de la posición de cada elemento simplemente agregando un desplazamiento a un valor base, es decir, la ubicación de memoria del primer elemento de la matriz (generalmente indicado por el nombre del arreglo). El valor base es el índice 0 y la diferencia entre los dos índices es el desplazamiento.




La imagen de arriba se puede ver como una vista de nivel superior de una escalera donde se encuentra en la base de la escalera. Cada elemento se puede identificar de forma única por su índice en la matriz.

- **Una Matrix** es una estructura similar a una tabla que consta de elementos dispuestos en un número fijo de filas y columnas. Todos los elementos pertenecen a un solo tipo de datos. Se puede acceder a los elementos de una matriz proporcionando índices de filas y columnas. La operación aritmética, suma, resta y multiplicación se puede realizar en matrices con las mismas dimensiones.

A continuación se muestra una matriz con 9 elementos.

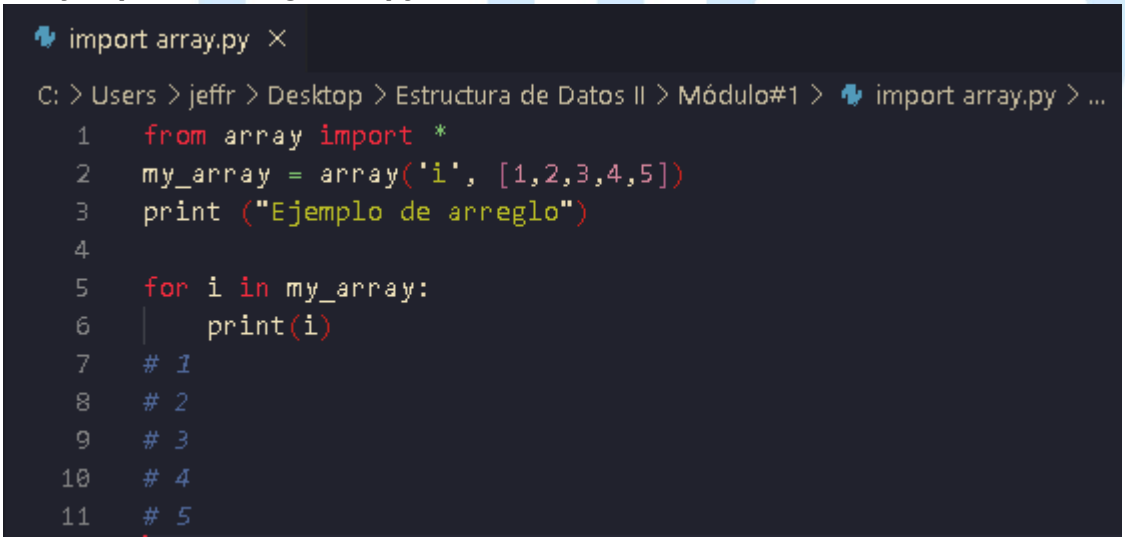



Esta Matriz [M] tiene 3 filas y 3 columnas. Se puede hacer referencia a cada elemento de la matriz [M] por su número de fila y columna. Por ejemplo, un 23 = 6.

**Cuadro de diferencia de arreglos y matrices:**

| Arreglos | Matrices |
|---|---|
| Los arreglos pueden contener dimensiones mayores o iguales a 1. | Matrices contiene 2 dimensiones en una tabla como estructura. |
| Array es una estructura de datos homogénea. | Matrix es también una estructura de datos homogénea. |
| Los elementos están organizados linealmente. | Los elementos están organizados bidemensionalmente. |
| Es un vector singular dispuesto en las dimensiones especificadas. | Se compone de múltiples vectores de igual longitud apilados juntos en una tabla |
| La función array() se puede usar para crear una matriz especificando que la tercera dimensión sea 1. | Sin embargo, la función matrix () se puede usar para crear una matriz bidimensional como máximo. |
| Los arreglos son un superconjunto de matrices. | Las matrices son un subconjunto, un caso especial de matriz donde las dimensiones son dos. |
| Conjunto limitado de operaciones basadas en colecciones. | Amplia gama de operaciones de recogida posibles. |
| En su mayoría, destinados al almacenamiento de datos. | En su mayoría, las matrices están destinadas a la transformación de datos. |

**Ejemplo de arreglo en python:**




**Ejemplo de Matriz en Python**

Python no tiene un tipo de dato incorporado para trabajar con matrices, sin embargo podemos tratar la matriz como una lista de listas, por ejemplo:

```
A = [[1, 4, 5],
     [-5, 8, 9]]
```

matriz.py

C: > Users > jeffr > Desktop > Estructura de Datos II > Módulo#1 > matriz.py > ...

```
1   print ("EJEMPLO DE UNA MATRIZ")
2
3   A = [[1,4,5,12],
4        [-5,8,9,0],
5        [-6,7,11,19]]
6
7   print ("A =",A)
8   print ("A[1] =", A[1]) # 2nd fila
9   print ("A[1][2] =",A[1][2])#3rd elemento de la segunda fila
10  print ("A[0][-1] =",A[0][-1])#último elemento de la primera fila
11
12  column =[]  #Lista vacia
13  for row in A:
14      column.append(row[2])
15
16  print ("3rd Columna =", column)
17
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

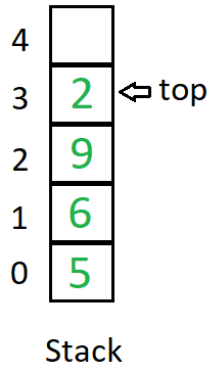
```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos

Prueba la nueva tecnología PowerShell multiplataforma h

PS C:\Users\jeffr> & C:/Python3/python.exe "c:/Users/je
EJEMPLO DE UNA MATRIZ
A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]
A[1] = [-5, 8, 9, 0]
A[1][2] = 9
A[0][-1] = 12
3rd Columna = [5, 9, 11]
```

## Las diferencias y dónde aplicar una pila o una cola

### Pila

Una pila es una estructura de datos lineal en la que los elementos se pueden insertar y eliminar solo desde un lado de la lista, llamado la parte superior . Una pila sigue el principio LIFO (Last In First Out), es decir, el último elemento insertado es el primero en salir. La inserción de un elemento en la pila se denomina operación de inserción y la eliminación de un elemento de la pila se denomina operación de extracción . En stack siempre hacemos un seguimiento del último elemento presente en la lista con un puntero llamado top .

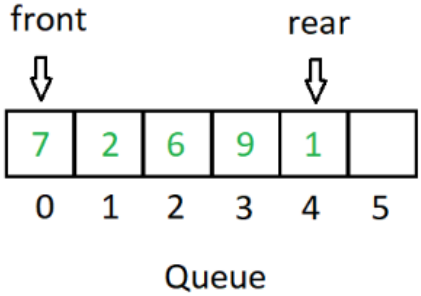
La representación esquemática de la pila se da a continuación:




### Cola:

Una cola es una estructura de datos lineal en la que los elementos se pueden insertar solo desde un lado de la lista llamado trasero , y los elementos se pueden eliminar solo desde el otro lado llamado frente . La estructura de datos de la cola sigue el principio **FIFO** (primero en entrar, primero en salir), es decir, el elemento insertado primero en la lista es el primer elemento que se elimina de la lista. La inserción de un elemento en una cola se denomina operación de puesta en cola y la eliminación de un elemento se denomina operación de eliminación de cola . En la cola siempre mantenemos dos punteros, uno que apunta al elemento que se insertó en primer lugar y que todavía está presente en la lista con el puntero delantero y el segundo puntero que apunta al elemento insertado en último lugar con el puntero trasero.

La representación esquemática de la cola se da a continuación:



Las pilas se utilizan en:

- ★ las aplicaciones basadas en la caché, como la aplicación abierta/usada recientemente aparecerá.
- ★ Lograr la operación de deshacer en los blocs de notas.
- ★ El botón de retroceso del navegador utiliza una pila.

Las colas se utilizan para:

- ★ Borrar/eliminar los datos, por ejemplo, los primeros datos insertados deben eliminarse primero.
- ★ En la vida real, los sistemas telefónicos de los centros de llamadas utilizarán colas, para retener a las personas que les llaman en un orden, hasta que un representante de servicio esté libre. Programación de la CPU, Programación del Disco. Cuando varios procesos requieren CPU al mismo tiempo, se utilizan varios algoritmos de programación de CPU que se implementan utilizando la estructura de datos de la cola.
- ★ En la cola de impresión
- ★ Búsqueda de la primera amplitud en un gráfico
- ★ Manejo de interrupciones en sistemas de tiempo real. Las interrupciones se manejan en el mismo orden en que llegan, el primero en llegar es el primero en ser atendido.

**Cuadro de diferencias**

| Pilas | Colas |
|---|---|
| Las pilas se basan en el principio LIFO, es decir, el último elemento insertado es el primer elemento que sale de la lista. | Las colas se basan en el principio FIFO, es decir, el elemento que se inserta primero es el primer elemento que sale de la lista. |
| La inserción y eliminación en las pilas se realiza solo desde un extremo de la lista llamado la parte superior. | La inserción y eliminación en las colas tiene lugar desde los extremos opuestos de la lista. La inserción tiene lugar al final de la lista y la eliminación tiene lugar al principio de la lista. |
| La operación de inserción se denomina operación de inserción. | La operación de inserción se denomina operación de puesta en cola. |
| La operación de eliminación se llama operación emergente. | La operación de eliminación se denomina operación de eliminación de cola. |
| En pilas mantenemos solo un puntero para acceder a la lista, llamado top, que siempre apunta al último elemento presente en la lista. | En las colas mantenemos dos punteros para acceder a la lista. El puntero delantero siempre apunta al primer elemento insertado en la lista y todavía está presente, y el puntero trasero siempre apunta al último elemento insertado. |
| Pila se usa para resolver problemas y funciona en recursividad. | La cola se utiliza para resolver problemas que tienen un procesamiento secuencial. |

**Ejemplo de pila en python:**

tarea.py > ...

```python
class Stack(object):
    "" "Pila" ""
    def __init__(self, item = []):
        self.item = []
        if len(item):
            for i in item:
                self.item.append(i)

    def push(self, item):
            self.item.append(item)

    def clear(self):
        del self.item

    def is_empty(self):
        return self.size() == 0

    def size(self):
        return len(self.item)

    def print(self):
        print(self.item)

    def top(self):
        return self.item[-1]

    def pop(self):
        data = self.top()
        self.item.pop()
        return data

print("Crear pila")
stack =  Stack([1,2,3])
stack.print()
print("Insertar elemento en la parte superior de la pila")
stack.push(4)
stack.print()
print("Determinar si la pila está vacía")
print(stack.is_empty())
print("Devuelve el número de elementos en la pila")
print(stack.size())
print("Artículo de vuelta a la parte superior de la pila")
print(stack.top())
print("Eliminar elemento en la parte superior de la pila")
stack.pop()
stack.print()
print("Pila vacía")
print(stack.clear())
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\jeffr\Documents\tarea> & C:/Python3/python.exe c:/Users/jeffr/Documents/tarea/tarea.py
Crear pila
[1, 2, 3]
Insertar elemento en la parte superior de la pila
[1, 2, 3, 4]
Determinar si la pila está vacía
False
Devuelve el número de elementos en la pila
4
Artículo de vuelta a la parte superior de la pila
4
Eliminar elemento en la parte superior de la pila
[1, 2, 3]
Pila vacía
None
PS C:\Users\jeffr\Documents\tarea>
```

**Ejemplo de cola en python:**

tarea.py > ...

```python
class Queue(object):
    "" "Cola de simulación" ""
    def __init__(self, item = []):
        self.item = []
        if len(item):
            for i in item:
                self.item.append(i)

    def enqueue(self, item):
        self.item.append(item)

    def dequeue(self):
        self.item.pop(0)

    def clear(self):
        del self.item

    def is_empty(self):
        return self.size() == 0

    def size(self):
        return len(self.item)

    def print(self):
        print(self.item)

print("Crear cola")
queue =  Queue([1,2,3])
queue.print()
print("Insertar elementos en la cola")
queue.enqueue(4)
queue.print()
```

```
print("Eliminar elemento de la cola")
queue.dequeue()
queue.print()
print("Determinar si la cola está vacía")
print(queue.is_empty())
print("Devuelve el número de elementos en la cola")
print(queue.size())
queue.print()
print("Cola vacía")
print(queue.clear())
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

PS C:\Users\jeffr\Documents\tarea> & C:/Python3/python.exe c:/Users/jeffr/Docum
Crear cola
[1, 2, 3]
Insertar elementos en la cola
[1, 2, 3, 4]
Eliminar elemento de la cola
[2, 3, 4]
Determinar si la cola está vacía
False
Devuelve el número de elementos en la cola
3
[2, 3, 4]
Cola vacía
None
PS C:\Users\jeffr\Documents\tarea>
```

**3. Buscar: que se necesita para realizar una estructura de lista.**

- **Lista**

Una lista es una estructura dinámica de datos que contiene una colección de elementos homogéneos (del mismo tipo) de manera que se establece entre ellos un orden. Es decir, cada elemento, menos el primero, tiene un predecesor, y cada elemento, menos el último, tiene un sucesor.

Las listas en Python son un tipo contenedor, compuesto, que se usan para almacenar conjuntos de elementos relacionados del mismo tipo o de tipos distintos.

Junto a las clases tuple, range y str, son uno de los tipos de secuencia en Python, con la particularidad de que son mutables. Esto último quiere decir que su contenido se puede modificar después de haber sido creada.

Para crear una lista en Python, simplemente hay que encerrar una secuencia de elementos separados por comas entre paréntesis cuadrados [].

Por ejemplo, para crear una lista con los números del 1 al 10 se haría del siguiente modo:

```
IDLE Shell 3.9.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numeros
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

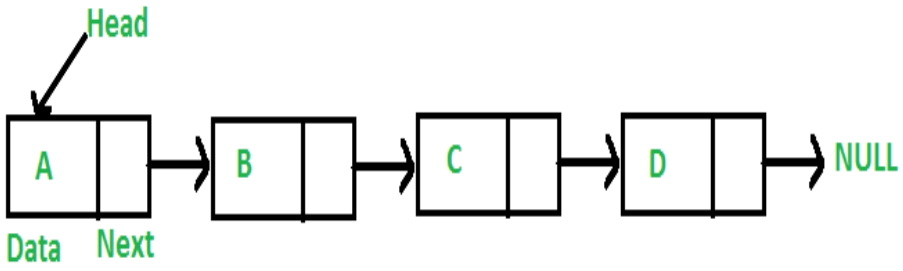
Las listas también se pueden crear usando el constructor de la clase, list(iterable). En este caso, el constructor crea una lista cuyos elementos son los mismos y están en el mismo orden que los ítems del iterable. El objeto iterable puede ser o una secuencia, un contenedor que soporte la iteración o un objeto iterador.

Por ejemplo, el tipo str también es un tipo secuencia. Si pasamos un string al constructor list() creará una lista cuyos elementos son cada uno de los caracteres de la cadena:

```
IDLE Shell 3.9.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> vocales = list('aeiou')
>>> vocales
['a', 'e', 'i', 'o', 'u']
```

- **Lista enlazada**

La lista enlazada es una estructura de datos lineal. A diferencia de las matrices, los elementos de la lista enlazada no se almacenan en una ubicación contigua; los elementos se vinculan mediante punteros.




Una lista enlazada se representa mediante un puntero al primer nodo de la lista enlazada. El primer nodo se llama cabeza. Si la lista enlazada está vacía, entonces el valor de la cabeza apunta a NULL.

Cada nodo en una lista consta de al menos dos partes:

1. datos (podemos almacenar números enteros, cadenas o cualquier tipo de datos).
2. Puntero (o referencia) al siguiente nodo (conecta un nodo con otro)

linkedlist.py

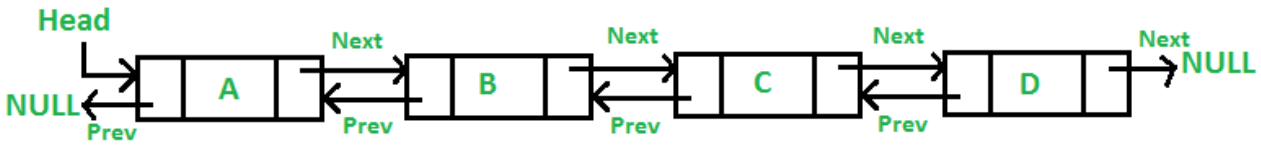
C: > Users > jeffr > Desktop > Estructura de Datos II > Módulo#1 > linkedlist.py > ...

```python
# A simple Python program for traversal of a linked list
# Node class
class Node:

    # Function to initialise the node object
    def __init__(self, data):
        self.data = data # Assign data
        self.next = None # Initialize next as null

# Linked List class contains a Node object
class LinkedList:

    # Function to initialize head
    def __init__(self):
        self.head = None

    # This function prints contents of linked list
    # starting from head
    def printList(self):
        temp = self.head
        while (temp):
            print (temp.data)
            temp = temp.next

# Code execution starts here
if __name__=='__main__':

    # Start with the empty list
    llist = LinkedList()
    llist.head = Node(1)
    second = Node(2)
    third = Node(3)

    llist.head.next = second; # Link first node with second
    second.next = third; # Link second node with the third node

    llist.printList()
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\jeffr> & C:/Python3/python.exe "c:/Users/jeffr/Desktop/Estructura
1
2
3
PS C:\Users\jeffr>
```

- **Lista doblemente enlazada**

Una lista enlazada doblemente ( DLL ) contiene un puntero adicional, normalmente llamado puntero anterior , junto con el puntero siguiente y los datos que están allí en la lista enlazada individualmente.




**Inserción**
Un nodo se puede agregar de cuatro maneras
**1)** Al frente de la DLL
**2)** Después de un nodo dado.
**3)** Al final de la DLL
**4)** Antes de un nodo dado.

- **Agregar un nodo al frente:** El nuevo nodo siempre se agrega antes del encabezado de la lista enlazada dada. Y el nodo recién agregado se convierte en el nuevo jefe de DLL.
- **Agregar un nodo después de un nodo dado:** Se nos da un puntero a un nodo como prev_node, y el nuevo nodo se inserta después del nodo dado.
- **Agregar un nodo al final:** El nuevo nodo siempre se agrega después del último nodo de la lista enlazada dada.
- **Agregue un nodo antes de un nodo dado:**

1. Compruebe si next_node es NULL o no. Si es NULL, regrese de la función porque no se puede agregar ningún nodo nuevo antes de NULL
2. Asigne memoria para el nuevo nodo, que se llame new_node
3. Establecer nuevo_nodo->datos = nuevos_datos

4. Establezca el puntero anterior de este new_node como el nodo anterior del next_node, new_node->prev = next_node->prev
5. Establezca el puntero anterior de next_node como new_node, next_node->prev = new_node
6. Establezca el siguiente puntero de este new_node como next_node, new_node->next = next_node;
7. Si el nodo anterior de new_node no es NULL, establezca el siguiente puntero de este nodo anterior como new_node, new_node->prev->next = new_node
8. De lo contrario, si la anterior de new_node es NULL, será el nuevo nodo principal. Entonces, haz (*head_ref) = new_node.

DoublyLinkedList.py

C: > Users > jeffr > Desktop > Estructura de Datos II > Módulo#1 > DoublyLinkedList.py > DoublyLink

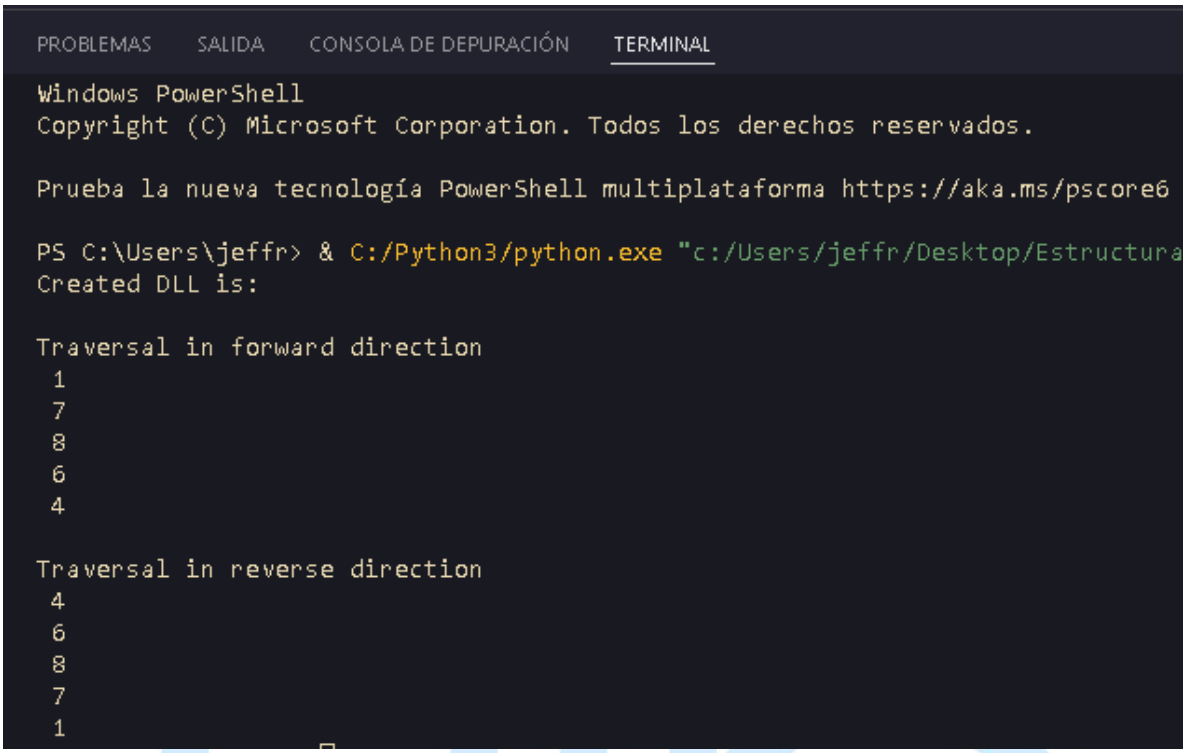
```python
# A complete working Python:program to demonstrate all insertion methods

# A linked list node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Class to create a Doubly Linked List
class DoublyLinkedList:

    # Constructor for empty Doubly Linked List
    def __init__(self):
        self.head = None

    # Given a reference to the head of a list and an
    # integer, inserts a new node on the front of list
    def push(self, new_data):

        # 1. Allocates node
        # 2. Put the data in it
        new_node = Node(new_data)
```

```python
        # 3. Make next of new node as head and
        # previous as None (already None)
        new_node.next = self.head

        # 4. change prev of head node to new_node
        if self.head is not None:
            self.head.prev = new_node
```

```
        # 5. move the head to point to the new node
        self.head = new_node

    # Given a node as prev_node, insert a new node after
    # the given node
    def insertAfter(self, prev_node, new_data):

        # 1. Check if the given prev_node is None
        if prev_node is None:
            print("the given previous node cannot be NULL")
            return

        # 2. allocate new node
        # 3. put in the data
        new_node = Node(new_data)

        # 4. Make net of new node as next of prev node
        new_node.next = prev_node.next

        # 5. Make prev_node as previous of new_node
        prev_node.next = new_node

        # 6. Make prev_node ass previous of new_node
        new_node.prev = prev_node

        # 7. Change previous of new_nodes's next node
        if new_node.next:
            new_node.next.prev = new_node

    # Given a reference to the head of DLL and integer,
    # appends a new node at the end
    def append(self, new_data):
```

```
        # 1. Allocates node
        # 2. Put in the data
        new_node = Node(new_data)

        # 3. This new node is going to be the last node,
        # so make next of it as None
        # (It already is initialized as None)

        # 4. If the Linked List is empty, then make the
        # new node as head
        if self.head is None:
            self.head = new_node
            return

        # 5. Else traverse till the last node
        last = self.head
        while last.next:
            last = last.next
```

```
87          # 6. Change the next of last node
88          last.next = new_node
89
90          # 7. Make last node as previous of new node
91          new_node.prev = last
92
93          return
94
95      # This function prints contents of linked list
96      # starting from the given node
97      def printList(self, node):
98
99          print("\nTraversal in forward direction")
100         while node:
101             print(" {}".format(node.data))
102             last = node
103             node = node.next
104
105         print("\nTraversal in reverse direction")
106         while last:
107             print(" {}".format(last.data))
108             last = last.prev
109
110 # Driver program to test above functions
111
112 # Start with empty list
113 llist = DoublyLinkedList()
114
115 # Insert 6. So the list becomes 6->None
116 llist.append(6)
118 # Insert 7 at the beginning.
119 # So linked list becomes 7->6->None
120 llist.push(7)
121
122 # Insert 1 at the beginning.
123 # So linked list becomes 1->7->6->None
124 llist.push(1)
125
126 # Insert 4 at the end.
127 # So linked list becomes 1->7->6->4->None
128 llist.append(4)
129
130 # Insert 8, after 7.
131 # So linked list becomes 1->7->8->6->4->None
132 llist.insertAfter(llist.head.next, 8)
133
134 print ("Created DLL is: ")
135 llist.printList(llist.head)
136
137 # This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```


PROBLEMAS
SALIDA
CONSOLA DE DEPURACIÓN
TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\jeffr> & C:/Python3/python.exe "c:/Users/jeffr/Desktop/Estructura
Created DLL is:
Traversal in forward direction
1
7
8
6
4
Traversal in reverse direction
4
6
8
7
1