TADs

Los tipos de datos abstractos son estructuras de datos que definen un conjunto de operaciones y restricciones para manipular los datos, sin especificar cómo se implementan internamente. Estos tipos de datos permiten abstraer la complejidad de los datos y enfocarse en las operaciones que se pueden realizar sobre ellos. En la vida real, un ejemplo de un tipo de dato abstracto es una lista de compras. La lista de compras es una estructura que tiene ciertas operaciones (agregar un elemento, eliminar un elemento, verificar si un elemento está en la lista, etc.), pero no importa cómo se almacenen los elementos de la lista o cómo se implementen estas operaciones.

> En Python, los tipos de datos abstractos se pueden implementar utilizando clases y objetos. Aquí hay algunas explicaciones extendidas de cómo implementar los tipos de datos nodos, lista enlazada, pila y cola en Python:
>
> > Nodo: Un nodo es un elemento básico en una estructura de datos enlazada, que contiene un valor y una referencia al siguiente nodo en la lista. En Python, se puede implementar un nodo como una clase que tiene dos atributos: valor y siguiente. El atributo valor almacena el valor del nodo, mientras que el atributo siguiente es una referencia al siguiente nodo en la lista. Por ejemplo:

In [ ]:

```python
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.siguiente = None
```

> > Lista enlazada: Una lista enlazada es una estructura de datos en la que los elementos están conectados por nodos, donde cada nodo tiene una referencia al siguiente nodo en la lista. En Python, se puede implementar una lista enlazada como una clase que tiene un atributo cabeza, que es una referencia al primer nodo en la lista. La lista enlazada también tiene varias operaciones, como agregar un elemento al final de la lista, eliminar un elemento de la lista y buscar un elemento en la lista. Por ejemplo:

In [ ]:

```
class ListaEnlazada:
    def __init__(self):
        self.cabeza = None

    def agregar(self, valor):
        nuevo_nodo = Nodo(valor)
        if self.cabeza is None:
            self.cabeza = nuevo_nodo
        else:
            nodo_actual = self.cabeza
            while nodo_actual.siguiente is not None:
                nodo_actual = nodo_actual.siguiente
            nodo_actual.siguiente = nuevo_nodo

    def eliminar(self, valor):
        if self.cabeza is None:
            return
        if self.cabeza.valor == valor:
            self.cabeza = self.cabeza.siguiente
            return
        nodo_actual = self.cabeza
        while nodo_actual.siguiente is not None:
            if nodo_actual.siguiente.valor == valor:
                nodo_actual.siguiente = nodo_actual.siguiente.siguiente
                return
            nodo_actual = nodo_actual.siguiente

    def buscar(self, valor):
        nodo_actual = self.cabeza
        while nodo_actual is not None:
            if nodo_actual.valor == valor:
                return True
            nodo_actual = nodo_actual.siguiente
        return False
```

Pila: Una pila es una estructura de datos en la que los elementos se agregan y se eliminan por el mismo extremo, llamado la cima de la pila. En Python, se puede implementar una pila como una clase que tiene un atributo cima, que es una referencia al último elemento agregado a la pila. La pila también tiene varias operaciones, como agregar un elemento a la cima de la pila, eliminar el elemento de la cima de la pila y verificar si la pila está vacía. Por ejemplo:

In [ ]:

```
class Pila:
    def __init__(self):
        self.cima = None

    def agregar(self, valor):
        nuevo_nodo = Nodo(valor)
        nuevo_nodo.siguiente = self.cima
        self.cima = nuevo_nodo

    def eliminar(self):
        if self.cima is None:
            return None
        valor = self.cima.valor
        self.cima = self.cima.siguiente
        return valor

    def esta_vacia(self):
        return self.cima is None
```

Cola: Una cola es una estructura de datos en la que los elementos se agregan por un extremo y se eliminan por el otro extremo, llamado la cola de la cola. En Python, se puede implementar una cola como una clase que tiene dos atributos: cabeza y cola, que son referencias al primer y último elemento de la cola, respectivamente. La cola también tiene varias operaciones, como agregar un elemento a la cola, eliminar el elemento de la cabeza de la cola y verificar si la cola está vacía. Por ejemplo:

In [ ]:

```python
class Cola:
    def __init__(self):
        self.cabeza = None
        self.cola = None

    def agregar(self, valor):
        nuevo_nodo = Nodo(valor)
        if self.cabeza is None:
            self.cabeza = nuevo_nodo
            self.cola = nuevo_nodo
        else:
            self.cola.siguiente = nuevo_nodo
            self.cola = nuevo_nodo

    def eliminar(self):
        if self.cabeza is None:
            return None
        valor = self.cabeza.valor
        self.cabeza = self.cabeza.siguiente
        if self.cabeza is None:
            self.cola = None
        return valor

    def esta_vacia(self):
        return self.cabeza is None
```