Estructura de datos:

Pila

Es un conjunto de elementos los cuales están apilado uno encima del otro, es una estructura que se rige bajo el principio de LIFO (last in-first out), es decir, el ultimo elemento al entrar es el primero en salir. Un elemento puede ser agregado en cualquier momento a una pila, pero solo se puede acceder o eliminar el elemento que esté en la cima o tope de la misma, como se observa en la figura 1.

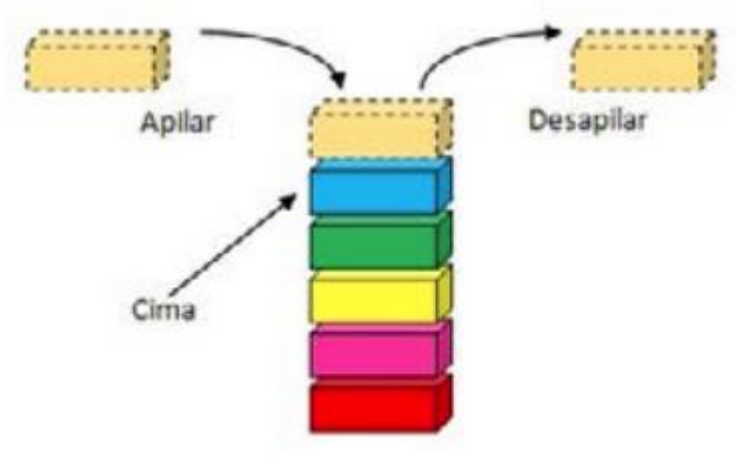


Figura 1. Funcionamiento de una pila

Esto implica que solo se puede acceder al elemento que está en la cima de la pila. La forma de acceder a los demás elementos de la colección es desapilar cada uno de estos, de a uno a la vez.

Algunos ejemplos donde implícitamente interactuamos con pilas en la vida cotidiana: los productos en una góndola, libros sobre una mesa, la ropa doblada, cajas en un depósito, platos o vasos en una alacena, etcétera. En cualquiera de estos casos para poder quitar o acceder al último elemento, es decir, el que está más abajo sobre la base de la pila– es necesario quitar los anteriores que están arriba, de lo contrario se caerían todos los elementos y se destruye la pila.

Si se observa la pila con un enfoque abstracto, es decir su modelo virtual, también se lo utiliza sin tenerlo en cuenta, por ejemplo, cuando se navega por internet el navegador trabaja con dos pilas que permiten ir a la dirección anterior, es decir volver a la página que estaba antes o ir hacia adelante a una página que ya se visitó; cuando se trabaja en un editor de texto existen los botones “hacer” y “deshacer”, esto permite al usuario volver a versiones anteriores o posteriores del texto que se está editando de manera rápida; lo mismo ocurre con los editores de imágenes, etcétera.

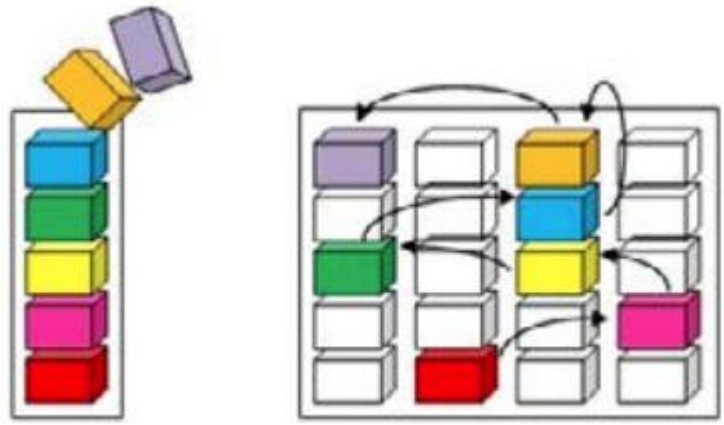
Por lo cual, una pila puede definirse de la siguiente manera: si consideramos su estructura y funcionamiento, es una estructura lineal dinámica de datos que no están ordenados y cuyas actividades de inserción y eliminación se realizan a través de un índice llamado cima o tope. Es importante remarcar que el orden de complejidad de las operaciones sobre la pila es de tiempo constante O(1) dado que solo se pueden agregar y quitar elementos desde la parte superior de la misma, por lo que no importa la cantidad de elementos que tenga. Por lo tanto, los elementos van a estar puestos según la inserción de cada uno.

¿Están realmente desordenados los datos?

En una pila por la naturaleza de su funcionamiento los datos no están ordenados siguiendo un criterio particular –salvo que se desarrolle un algoritmo que se encargue de ordenarlos–, pero si consideramos el orden de inserción de los elementos, se puede decir que la pila se encuentra ordenada por dicho criterio.

¿Como se lo puede representar?

A la Pila se lo puede representar en un Array, pero el problema es que el array no es dinámico *(es estático),* en algún momento se llena y no tengo más lugar. Pero, por otro lado, si usamos una implementación dinámica –a la derecha de la figura– con nodos enlazados disponemos de toda la memoria para crear nodos por lo cual es muy poco probable que se produzca un desbordamiento.



Para manejar esta estructura de dato, se va a tener que necesitar 5 funcionalidades, pero la última funcionalidad no es tan importante ya que me permite conocer cuántos elementos tengo dentro:

1. Apilar (pila, elemento). Agrega el elemento sobre la cima de la pila;
2. Desapilar (pila). Elimina y devuelve el elemento almacenado en la cima de la pila;
3. Pila_vacia (pila). Devuelve verdadero (true) si la pila no contiene elementos;
4. Cima (pila). Devuelve el valor del elemento que está almacenado en la cima de la pila, pero sin eliminarlo;
5. Tamaño(pila). Devuelve la cantidad de elementos en la pila.

#Se puede trabajar con distintos tipos de datos en una pila (*en Python*), pero siempre normalmente cuando estamos trabajando con gestión de datos es muy difícil ver con distintos tipos de datos, pero lo normal es que sean iguales.

En esta estructura de datos vamos a utilizar POO

Anotaciones:

Las clases para poder usarlas debemos de crearlas (*un objeto seria*), debemos de instanciar.

¿Cuál es la diferencia entre objeto y clase?

El **objeto** es la instancia de la clase, es decir, es algo que tengo en memoria y tiene datos. En cambio, la **clase** es simplemente una definición no tiene ejecución ni valor, entonces cuando yo creo una clase, esta se transforma en un objeto que va a estar en memoria, es decir, es una variable y acepta valores con sus *atributos* y puede ejecutar *funciones*.

Otra de las cuestiones que tiene una **clase**, es que por las buenas prácticas dicen que los *atributos* deberían ser **privados**, y la forma de poder acceder a ellos o modificarlos es a través de los *métodos/funciones* que son **públicas**. Los *métodos/funciones* es la interfaz mediante la cual yo me puedo conectar o puedo usar esa **clase**.
Y así para poder trabajar fuera de la **clase** lo que hago es poner el nombre del objeto, acompañado de un punto y el nombre de ese método/función; por ejemplo: pila.elementos[].

•*Self*: hace referencia al objeto y no a la clase.

Atributos privados en Python:
Para poder poner atributos privados en Python se hace con doble guion bajo delante de el atributo, por ejemplo: __elementos= [1,2,3].

Pasos para hacer una PILA:
Debemos definir un constructor, ya que todas las **clases** deben tener por lo menos un constructor, en Python el constructuctor debe de llamarse __init__. Y solo puedo tener uno. Pero si no tenemos un constructor definido, por más que tengamos atributos definidos y el programa ande, es porque existe un árbol de herencia y todas las clases que existan van a ser hijas de la clase **SuperClase**, esta va a ser padre o madre de las clases “menores”, en donde esta SuperClase tiene un constructor genérico que es sin atributos, entonces si cualquier clase que yo creo no tiene *constructo*r va a empezar a subir en el árbol de herencia hasta que algunas de las clases en el camino tenga un método constructor sin atributos y va a ejecutar ese mismo, por eso es que funciona.

El archivo de la clase me sirve para que después ocupe esos métodos en otro archivo, entonces me quedaría conformado de la siguiente manera:

1-**__init__**: constructor de la clase.

2-**push (***Apilar***):** Agrega el elemento sobre la cima de la pila.

3- **pop (***Desapilar***)**: Devuelve verdadero (true) si la pila no contiene elementos.

3- **size (***Tamaño***):** Devuelve la cantidad de elementos en la pila.

4- **on_top(***cima***):** Devuelve el valor del elemento que está almacenado en la cima de la pila, pero sin eliminarlo.

5- **show (***mostrar***):** muestra toda la pila.

```python
from typing import Any, Optional

class Stack:

    def __init__(self):
        self.__elements = []


    def push(self, value: Any) -> None:
        self.__elements.append(value)

    def pop(self) -> Optional[Any]:
        return (
            self.__elements.pop()
            if self.__elements #VERIFICA SI TIENE ELEMENTOS O NO
            else None
        )

    def size(self) -> int:
        return len(self.__elements)

    def on_top(self) -> Optional[Any]:
        return (
            self.__elements[-1] #OBJETO QUE ESTA EN LA CIMA
            if self.__elements
            else None
        )

    def show(self):
        aux_stack = Stack()
        while self.size() > 0:
            value = self.pop()
            print(value)
            aux_stack.push(value)

        while aux_stack.size() > 0:
            self.push(aux_stack.pop())
```