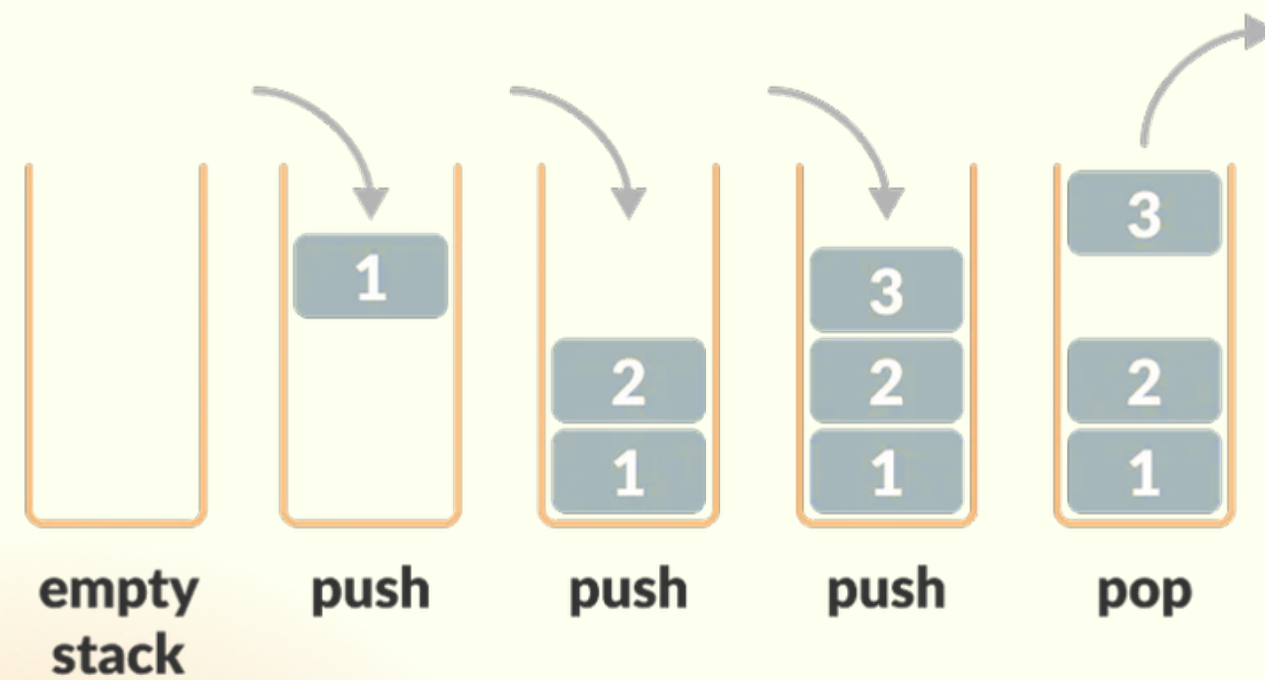
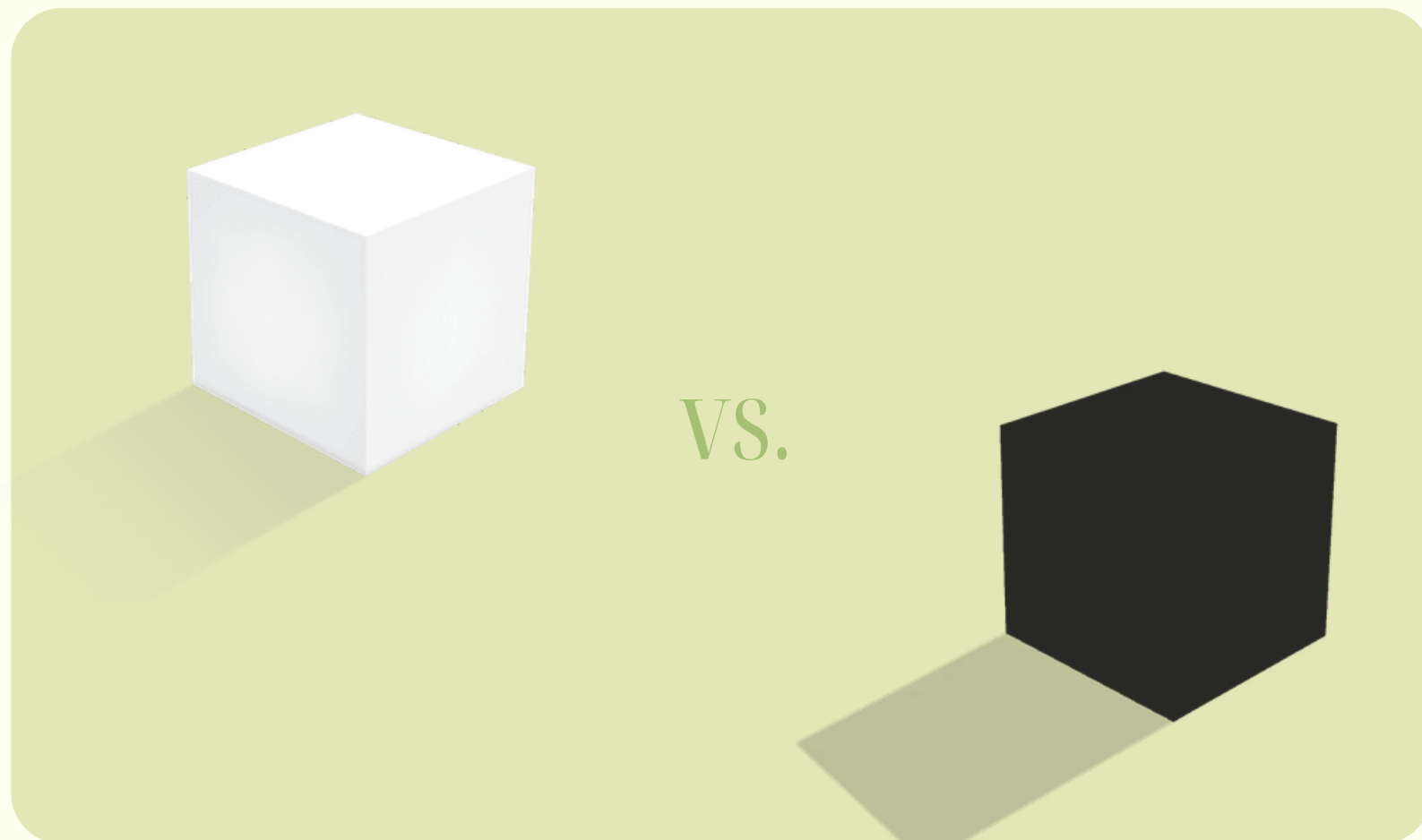


# TDA Pila



# ¿Qué es un TDA?

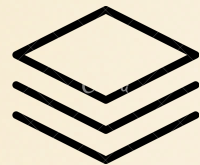


La **caja negra**. "No me importa cómo está hecho, me importa **lo qué hace**."





Algunos ejemplos de un comportamiento de un TDA Pila pueden ser un tubo de papas pringles, una pila de platos, una pila de libros, bandeja de entrada de correos,...



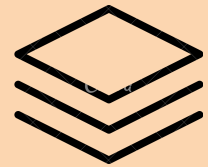
En esencia, una pila es un contenedor de elementos donde toda la acción ocurre en un solo lugar: el tope. Esta restricción en el acceso es lo que le da su poder y su comportamiento característico.

# TDA Pila

## ¿Qué es?

Una **pila** es una colección de elementos en la que pueden insertarse y eliminarse por un **extremo**, denominado **tope**, sus elementos.

# PILA : Estructura LIFO



LIFO significa Last-In, First-Out, que en español se traduce como "**Último en Entrar, Primero en Salir**".

Este es el principio que define a una pila y la diferencia de otras estructuras de datos.



# TDA Pila

Normalmente existe un conjunto de operaciones que se puede realizar sobre un tda.

¿Estas operaciones son estándares? No.

Existe un pequeño conjunto que siempre debería estar y después las mismas varían según las necesidades del programador.

Al conjunto que siempre debería estar se lo denominará:  
**conjunto mínimo de operaciones.**

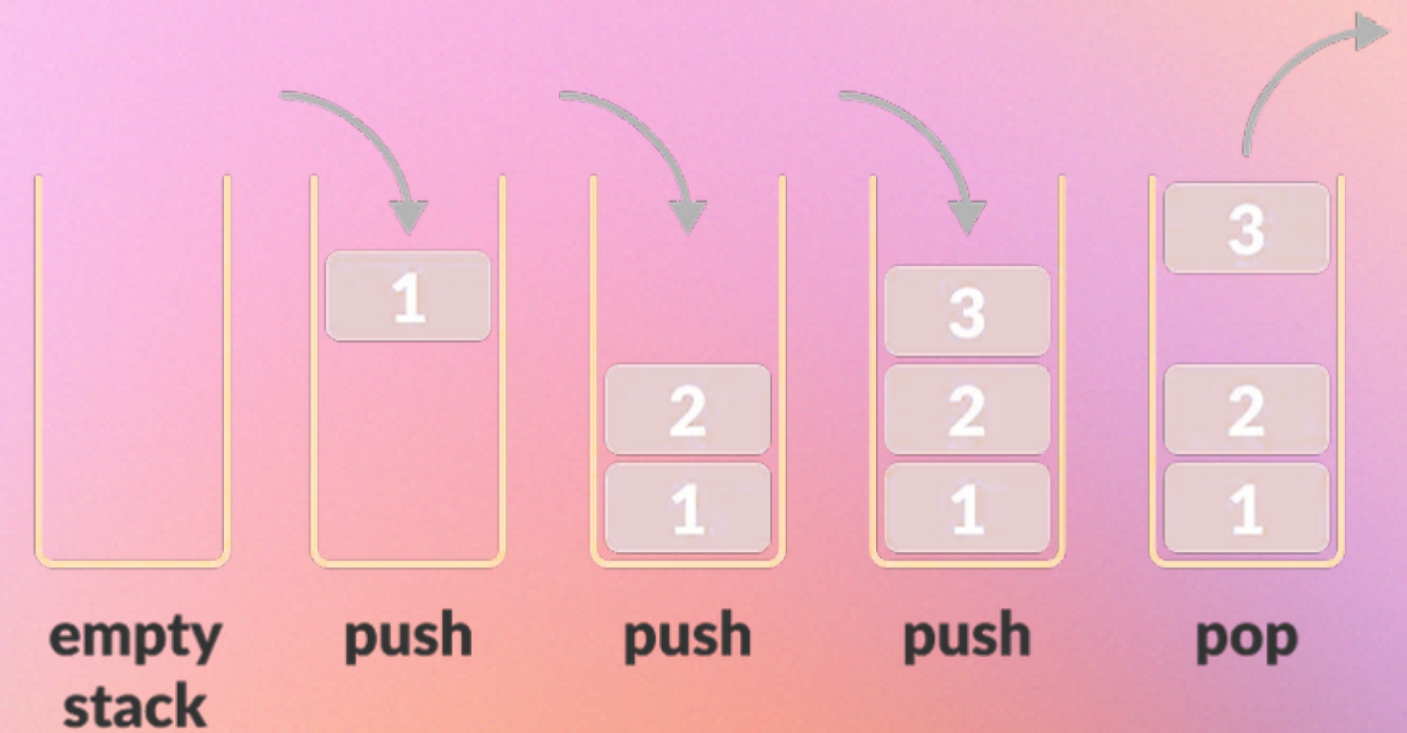


# TDA Pila

## Conjunto mínimo de operaciones

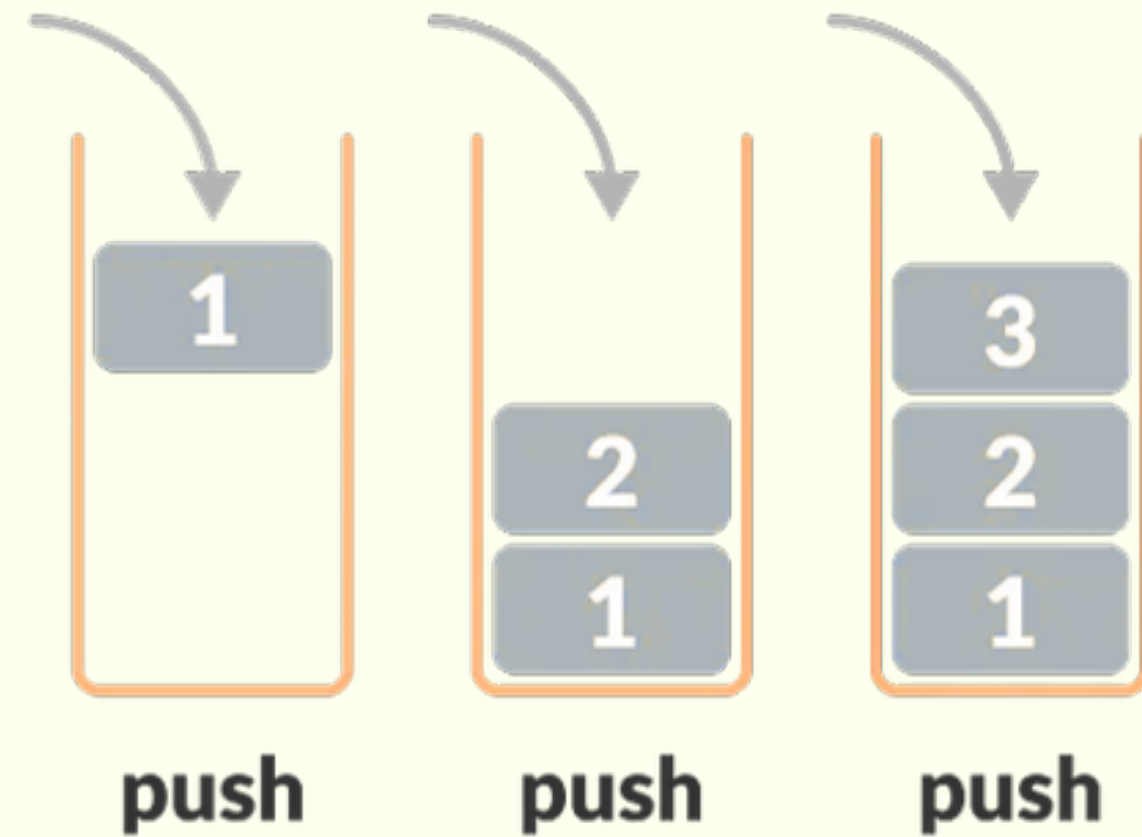
Para el caso de una pila el conjunto mínimo de operaciones es:

- crear (create)
- poner (push)
- sacar (pop)
- tope (top)
- esta\_vacia (is\_empty)
- destruir (destroy)



# TDA Pila: **apilar()**

insertar() / push()



# TDA Pila: apilar()

## insertar()

Esta operación pone un elemento en la pila por el tope de la misma, haciendo que el **tope de la pila pase a ser el nuevo elemento introducido**.

Valor:

Crear / Vaciar

Apilar

Desapilar

Tope

¿Está Vacía?

Destruir

▶ Auto Ejecutar

⏮ Paso Previo

Siguiente Paso ⏭

■ Cancelar

TDA Creado / Vaciado correctamente.

Crear / Vaciar

# TDA Pila: apilar()

insertar()

Esta operación pone un elemento en la pila por el tope de la misma, haciendo que el **tope de la pila pase a ser el nuevo elemento introducido**.

Valor: 7 | Crear / Vaciar | Apilar | Desapilar | Tope | ¿Está Vacía? | Destruir

**Apilar** `apilar(7)`

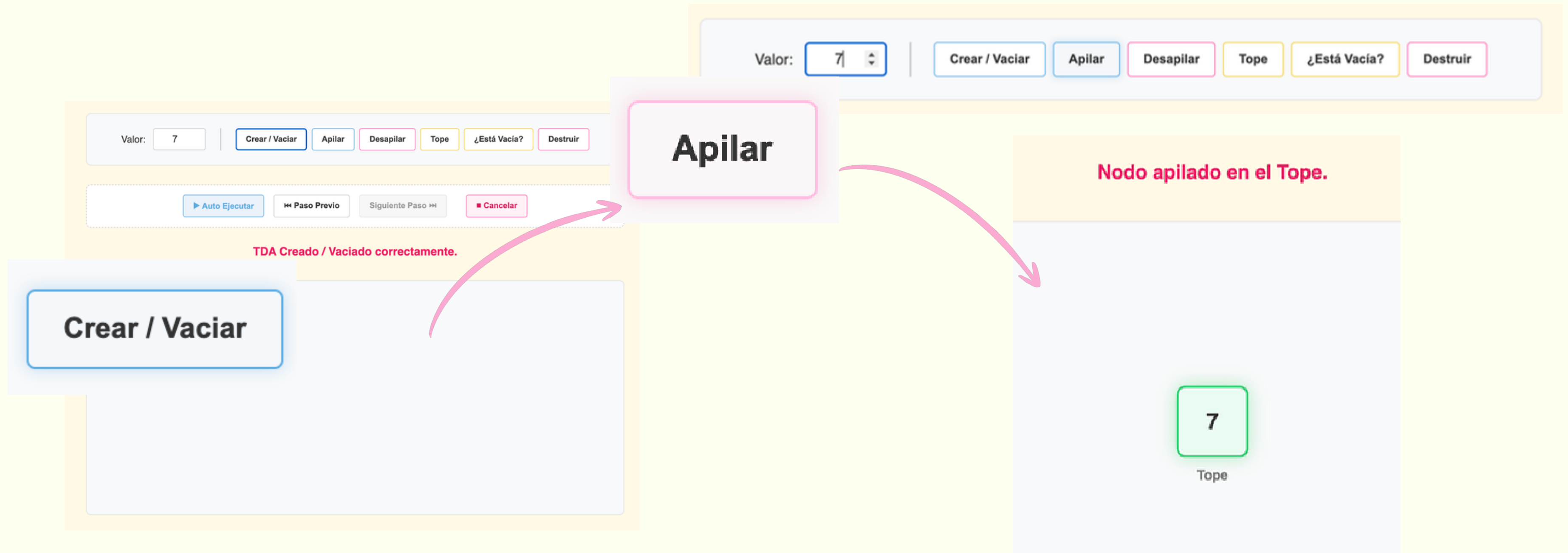
Crear / Vaciar

TDA Creado / Vaciado correctamente.

# TDA Pila: apilar()

insertar()

Esta operación pone un elemento en la pila por el tope de la misma, haciendo que el **tope de la pila pase a ser el nuevo elemento introducido**.



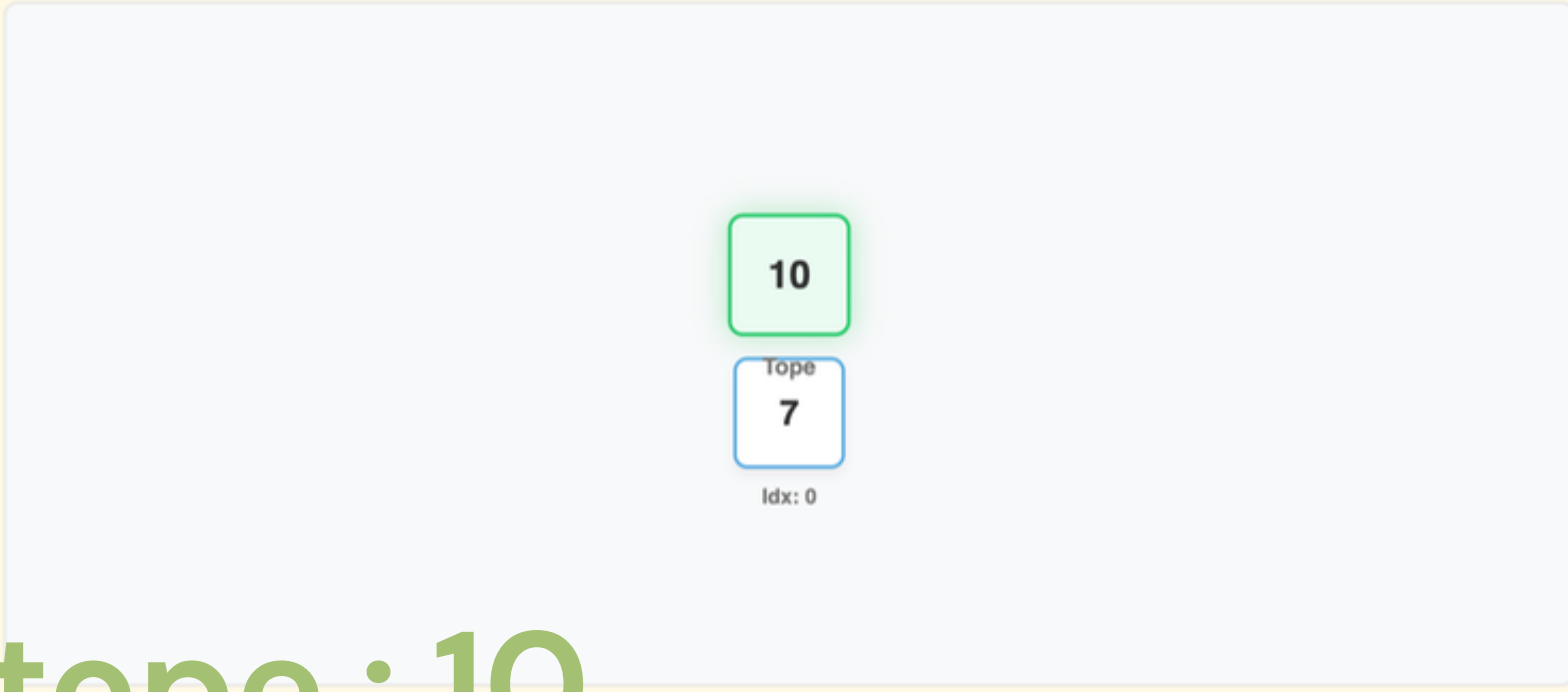
# TDA Pila: apilar()

## insertar()

Valor: 10 | Crear / Vaciar Apilar Desapilar Tope ¿Está Vacía? Destruir

Auto Ejecutar Paso Previo Siguiendo Paso Cancelar

Nodo apilado en el Tope.

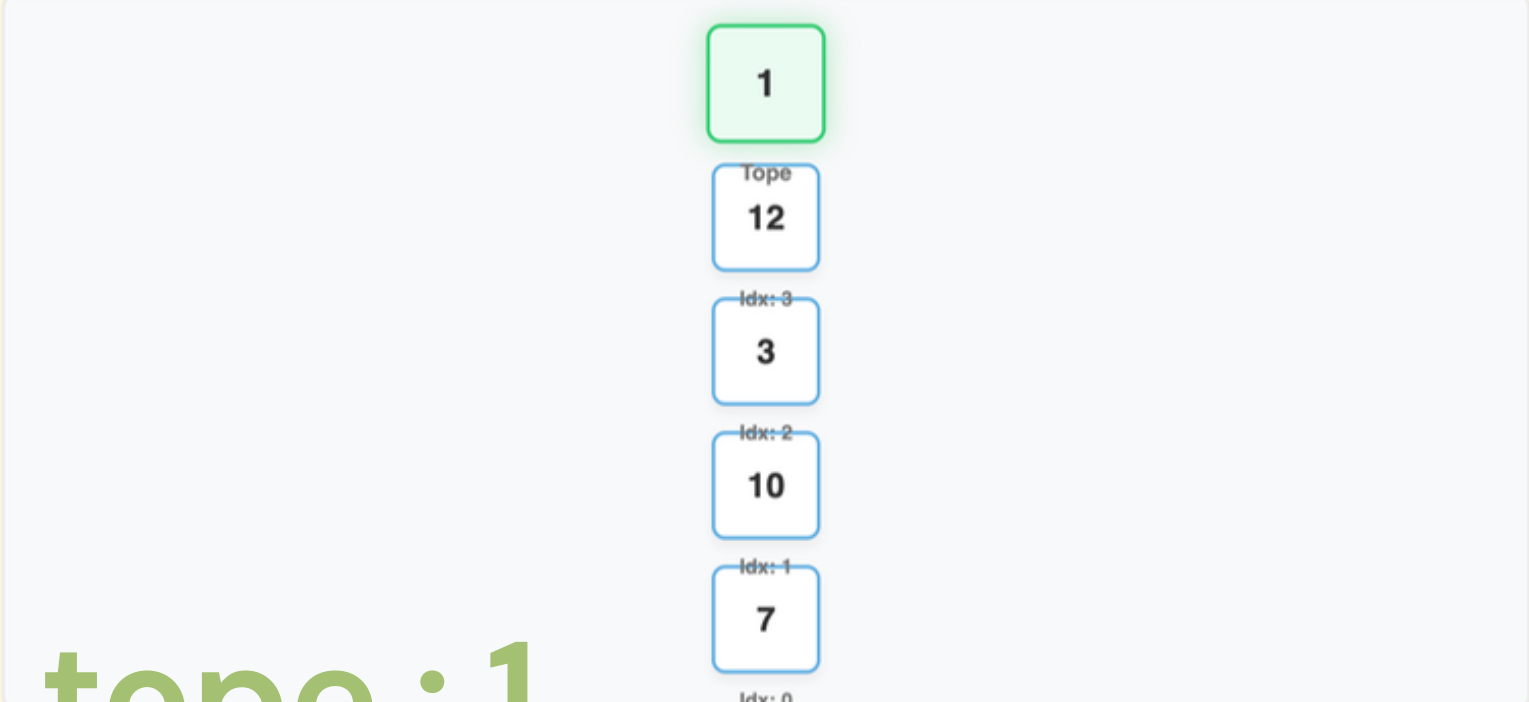


tope : 10

Valor: 1 | Crear / Vaciar Apilar Desapilar Tope ¿Está Vacía? Destruir

Auto Ejecutar Paso Previo Siguiendo Paso Cancelar

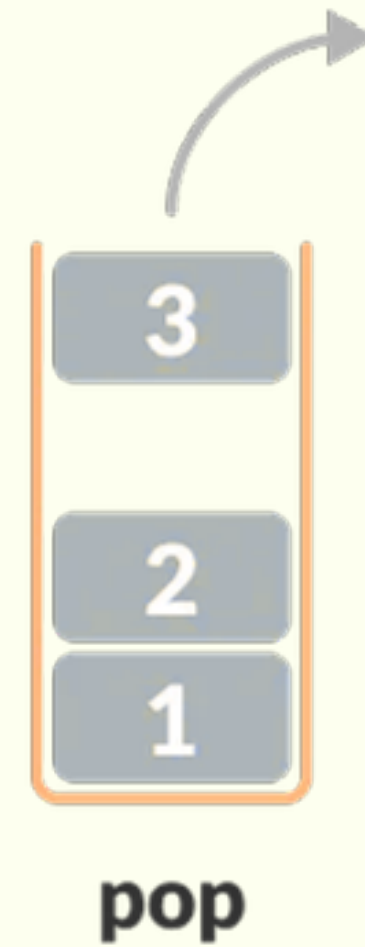
Nodo apilado en el Tope.



tope : 1

# TDA Pila: **desapilar()**

**eliminar()**

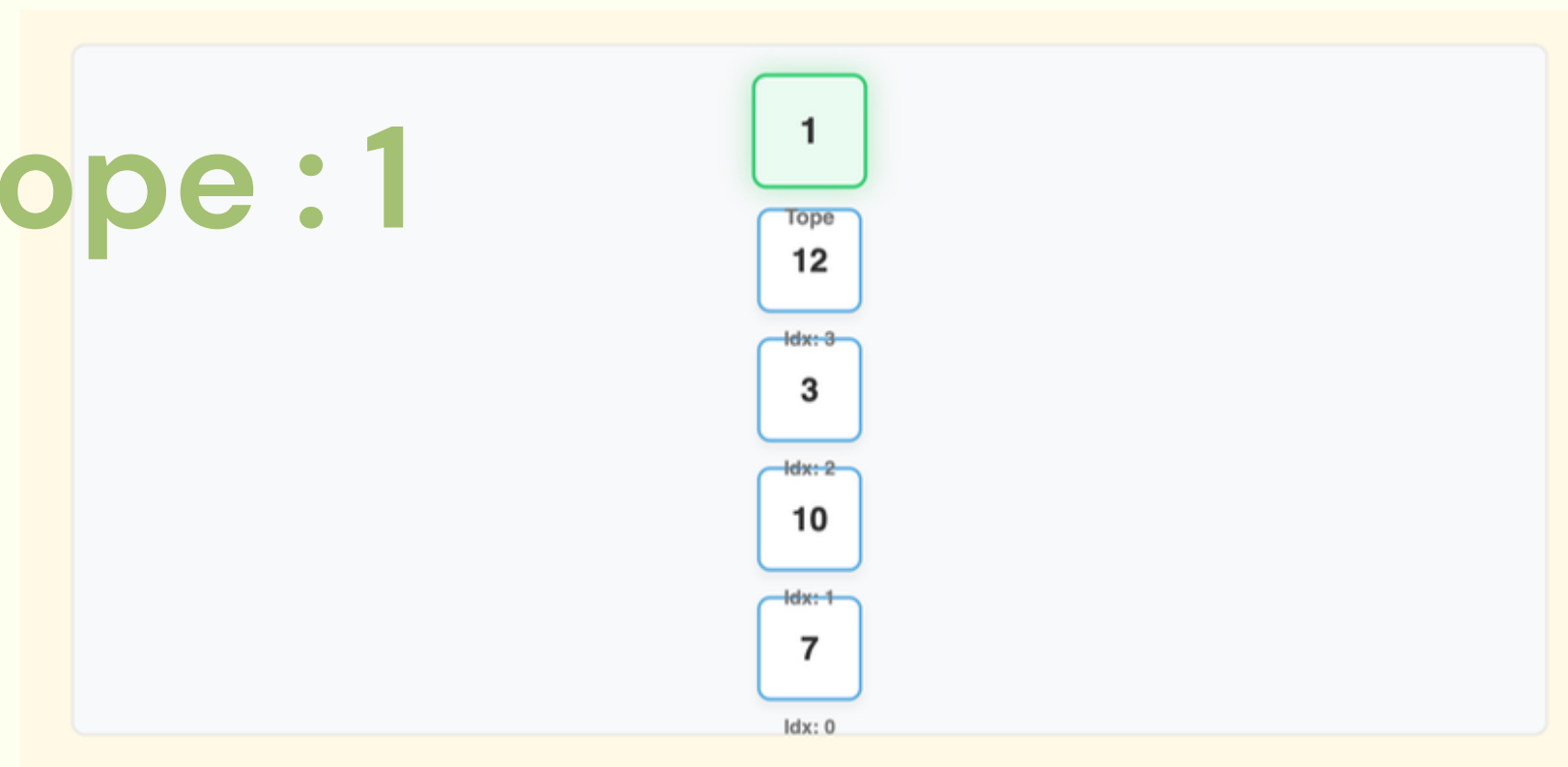


# TDA Pila: desapilar()

## eliminar()

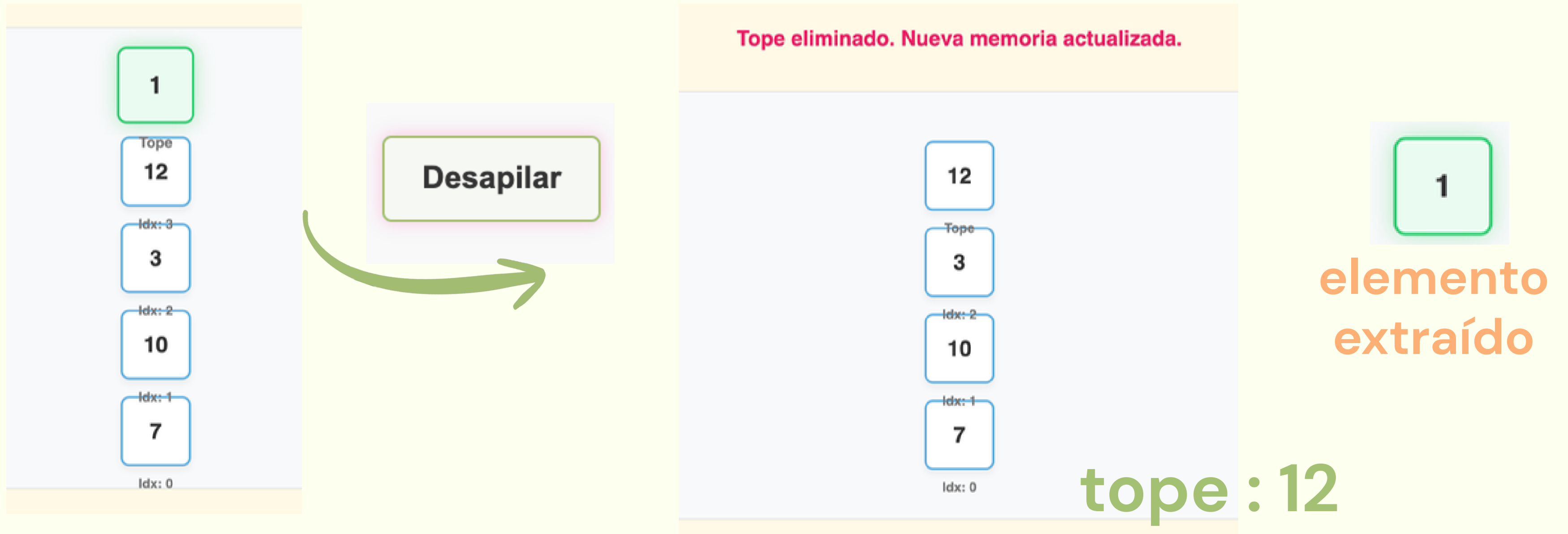
Esta operación **retira el elemento del tope de la pila** y mueve el tope de la pila al elemento anterior al extraído, si el elemento extraído es el último la pila queda vacía:

tope : 1



# TDA Pila: desapilar()

eliminar()



# TDA Pila: desapilar()

eliminar()

Valor:  |

**Tope eliminado. Nueva memoria actualizada.**

7

Tope

**Desapilar**

**Tope eliminado. Nueva memoria actualizada.**

# TDA Pila

- **crear** (create): Esta operación se ocupa de crear y reservar todos los recursos iniciales que se utilizarán para la pila.
- **tope** (top): Esta operación permite observar el valor del tope de la pila.
- **esta\_vacia** (is\_empty): Esta operación permite determinar si una pila tiene o no tiene elementos a través de un valor de verdad.
- **destruir** (destroy): Esta operación se ocupa de liberar y limpiar todos los recursos que se utilizan para la creación de una pila.

# ¿Cuándo usar una pila?

- **Hacer/ Deshacer**
- **Orden Inverso**
- **Análisis de expresiones**
- **Control de funciones (Call Stack)**

# Pros

- Simples de implementar.
- Optimización: Operaciones rápidas.
- Controlan el orden de manera natural:
  - Orden inverso
  - Recursividad
  - Historial

# Contras

- Acceso limitado de los elementos (**tope**).
- Implementación suele implicar un tamaño fijo: **stack overflow**.
- No sirve para problemas en los que necesitás acceso aleatorio, ordenar, buscar por posición, etc.

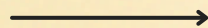
¿ Preguntas ?

# Ejercicios



Homero tiene un tubo con donas que compró en el kwik e mart y Marge quiere saber en qué orden las comería , sabiendo que Homero siempre agarra la dona de arriba del tubo.

Dado un paquete de donas, implementar una función que imprima el orden en el cual Homero comerá las donas. El paquete debe quedar vacío al final (Marge ya sabe que Homero nunca dejaría unas donas por la mitad).



Bart está armando su álbum de figuritas del mundial y quiere guardar sus figuritas en una pila.

Implementar una función que solicite al usuario ingresar nombres de jugadores y los apile. La carga finaliza cuando se ingresa la palabra "FIN".

