uc3m | Universidad Carlos III de Madrid

OpenCourseWare
Grado Ingeniería Informática
Estructura de Datos y Algoritmos

Tema 2 Tipos Abstractos de Datos Lineales: 2.1. Pilas y Colas

Profesores EDA

Objetivos

- Conocer los tipos abstractos de Pila y Cola.
- Ser capaz de desarrollar implementaciones de ambos TADs basadas en arrays (listas de Python).
- Ser capaz de resolver problemas típicos de Computación utilizando las estructuras lineales de pilas y colas.

Índice

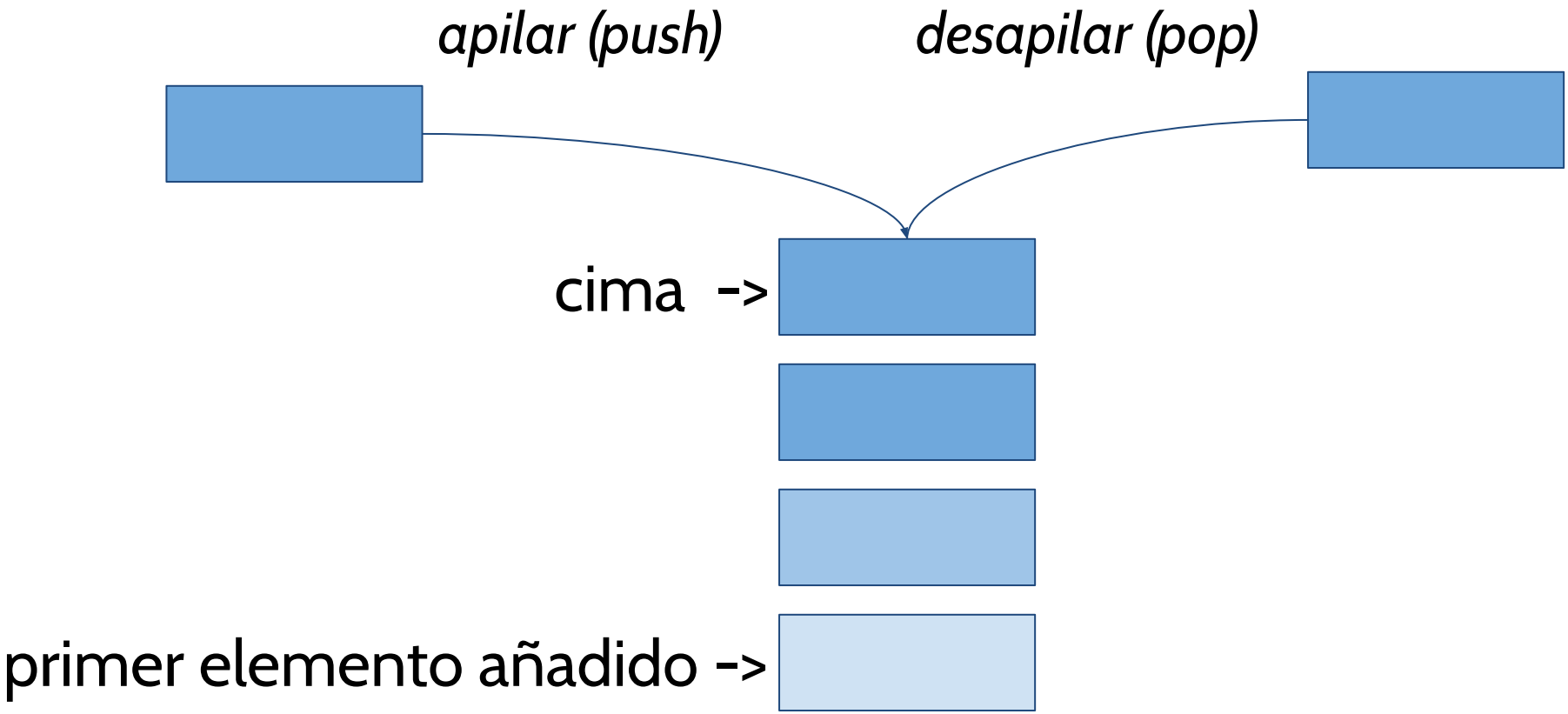
- TAD Pila
- TAD Cola
- TAD Lista
  - Implementación basada en lista simplemente enlazada
  - Implementación basada en lista doblemente enlazada.

¿Qué es un TAD lineal?

- Representa una **secuencia** de elementos de un mismo tipo.
- Ejemplos:
  - Nombres de personas: *Mary, John, Paul,* ...
  - Números primos: *1,2,3,5,7,11,13,17,19,...*
  - Objetos (instancias) de la clase MyComplex
  - Objetos (instancias) de la clase CreditCard

TAD Pila (Stack)

- Principio **LIFO (last-in, first-out)**.



TAD Pila (Stack)

- TAD lineal que permite almacenar y recuperar datos siguiendo el modo de acceso LIFO (last-in, first-out).
- Operaciones:
  - **Pila()** (en python será __init__): crea una pila vacía.
  - **apilar(e)**: añade un elemento, e, en la cima de la pila. No devuelve ningún valor.
  - **desapilar()**: borra y devuelve el elemento que está en la cima de la pila. Si la pila está vacía, se muestra un mensaje de error.

TAD Pila (Stack)

- **cima()**: devuelve el elemento que está en la cima de la pila, sin borrarlo. Es decir, la pila no es modificada. Si la pila está vacía, se muestra un mensaje de error.
- **tamaño**() (en python, __len__): devuelve el número de elementos que contiene la pila.
- **está_vacía**(): devuelve cierto si la pila está vacía, y falso en otro caso.

Implementación TAD Pila

- ¿Conoces alguna estructura de Python que nos permita fácilmente implementar una pila?
  - Debe permitir almacenar y recuperar datos siguiendo el principio LIFO (last-in, first-out).
  - Debe permitir implementar las operaciones del TAD Pila (apilar, desapilar, cima, etc).

Implementación del TAD Pila

- Una lista de Python proporciona métodos (append, pop, len, etc) que permiten fácilmente implementar una pila.
- Es una **implementación basada en arrays**.

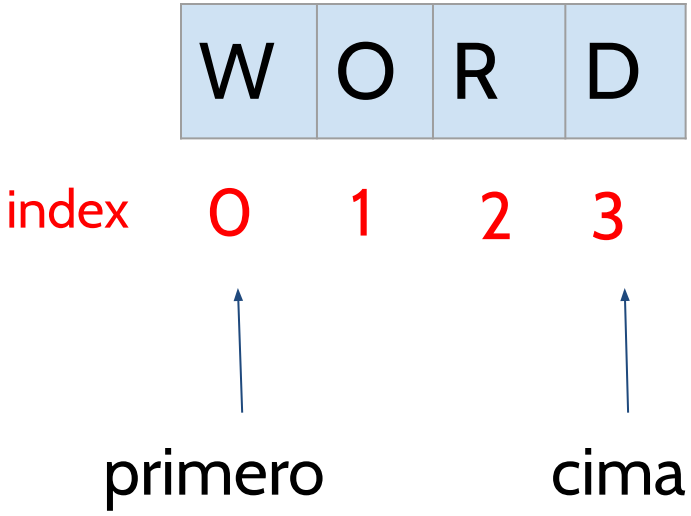
Implementación del TAD Pila

¿Dónde almacenamos la cima?, es decir, ¿En qué extremo de la lista debemos almacenar la cima de la pila? Dos opciones:

- **Opción 1:** La cima se almacenará siempre en la última posición del array (lista de Python).
- **Opción 2:** La cima se almacenará siempre en la posición 0 del array (lista de Python).

Implementación del TAD Pila

Opción 1:

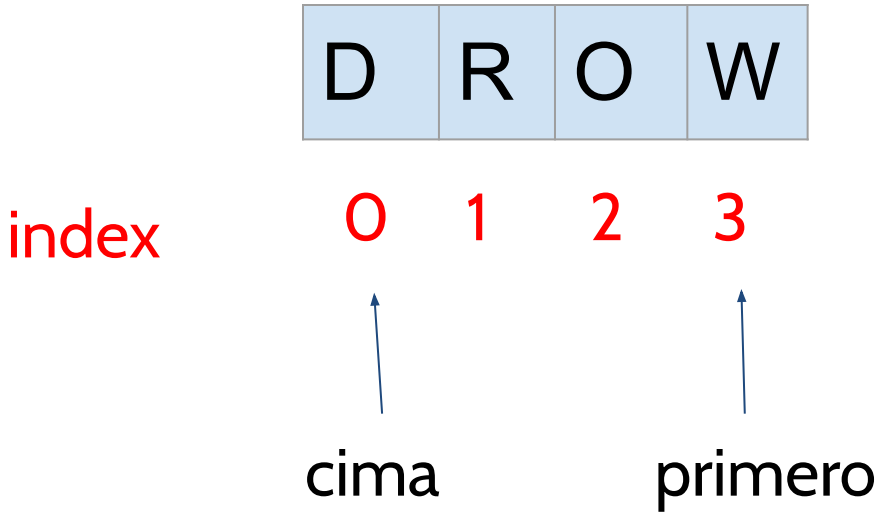


Implementación del TAD Pila

Opción 1: La **cima** es el último elemento de la lista.

Implementación del TAD Pila

Opción 2: la cima se almacena en la primera posición de la lista.



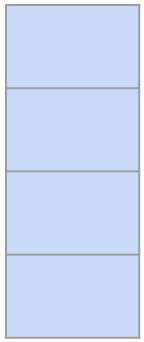
Implementación del TAD Pila

Opción 2: La **cima** es el primer elemento de la lista.

Resolución de problemas usando pilas

- Pilas son útiles para invertir datos. Por ejemplo, podemos usar una pila para **invertir una cadena de texto**.

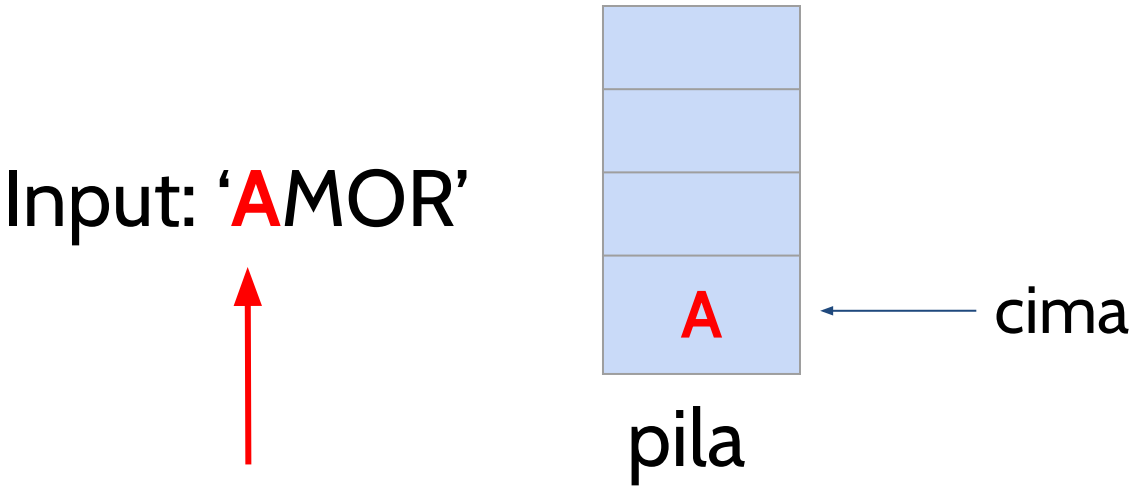
Input: 'AMOR'



pila

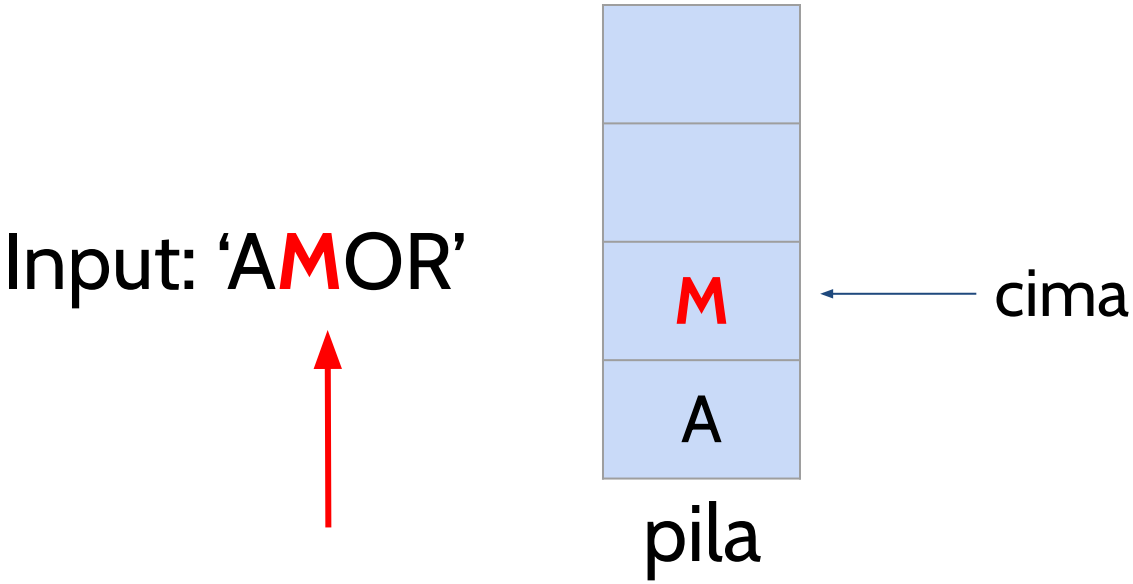
Resolución de problemas usando pilas

- ¿Cómo **invertir una cadena de texto**?
  - Recorremos la cadena de texto, carácter a carácter.
  - Cada carácter es añadido a la pila (apilar).



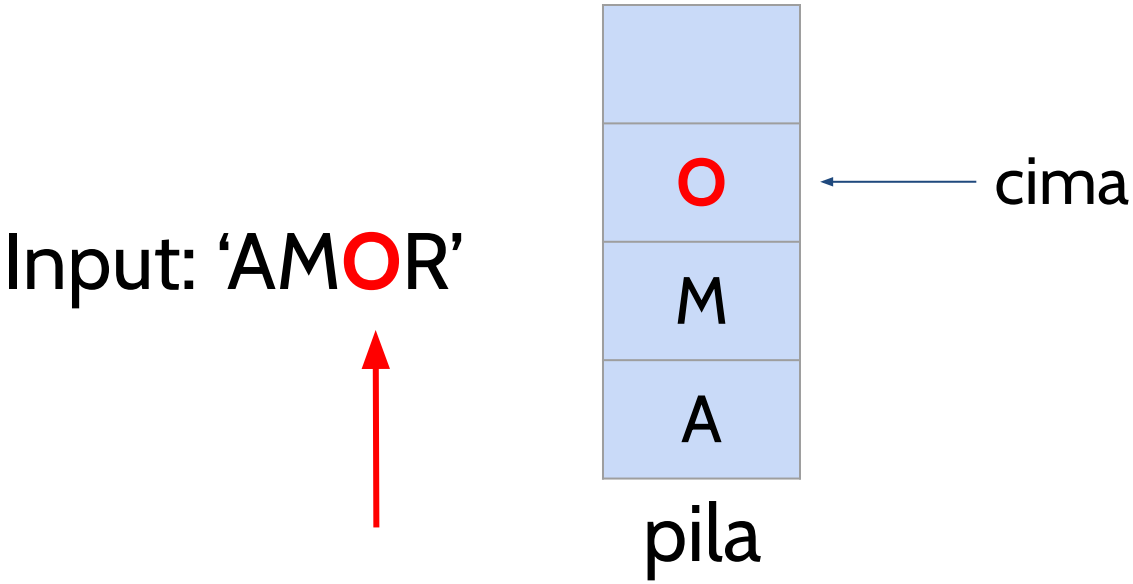
Resolución de problemas usando pilas

- ¿Cómo **invertir una cadena de texto**?
  - Recorremos la cadena de texto, carácter a carácter.
  - Cada carácter es añadido a la pila (apilar).



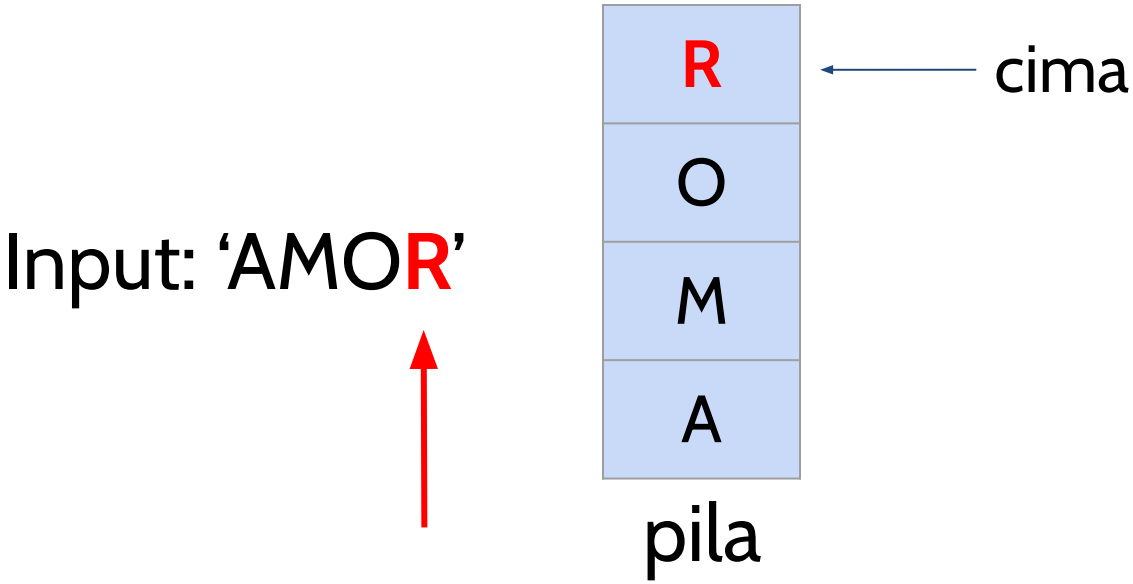
Resolución de problemas usando pilas

- ¿Cómo **invertir una cadena de texto**?
  - Recorremos la cadena de texto, carácter a carácter.
  - Cada carácter es añadido a la pila (apilar).



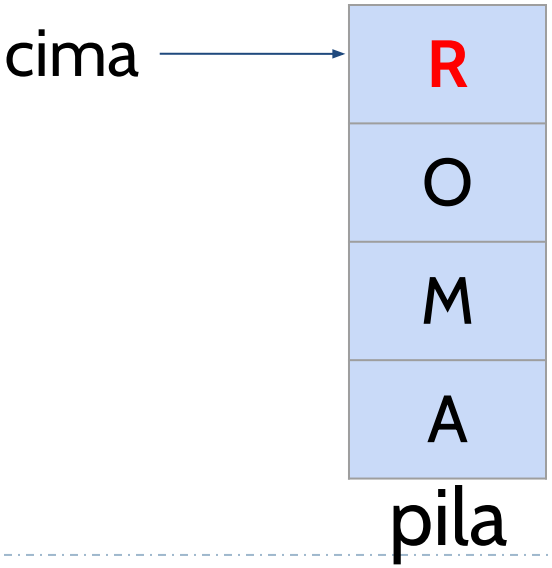
Resolución de problemas usando pilas

- ¿Cómo **invertir una cadena de texto**?
  - Recorremos la cadena de texto, carácter a carácter.
  - Cada carácter es añadido a la pila (apilar).



Resolución de problemas usando pilas

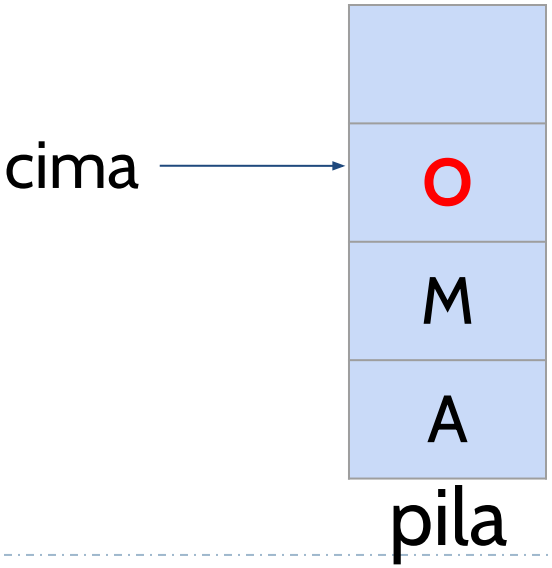
- ¿Cómo **invertir una cadena de texto**?
  - Una vez apilados todos los carácteres, desapilamos cada carácter de la pila, hasta que quede vacía.
  - Cada carácter desapilado se concatena al final de la cadena que devolveremos como salida.



Output:

Resolución de problemas usando pilas

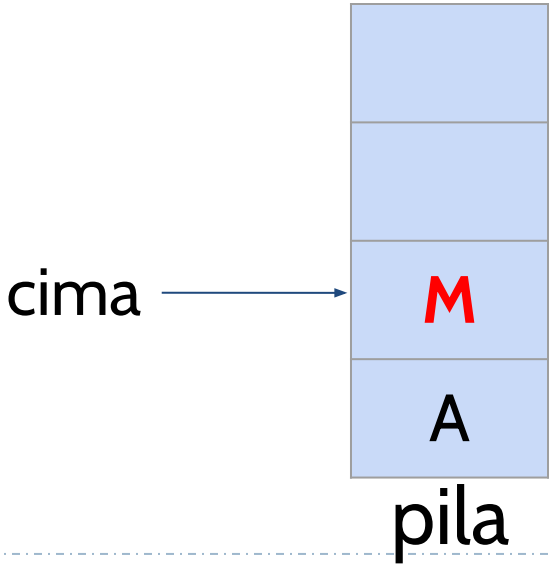
- ¿Cómo **invertir una cadena de texto**?
  - Una vez apilados todos los carácteres, desapilamos cada carácter de la pila, hasta que quede vacía.
  - Cada carácter desapilado se concatena al final de la cadena que devolveremos como salida.



Output: **R**

Resolución de problemas usando pilas

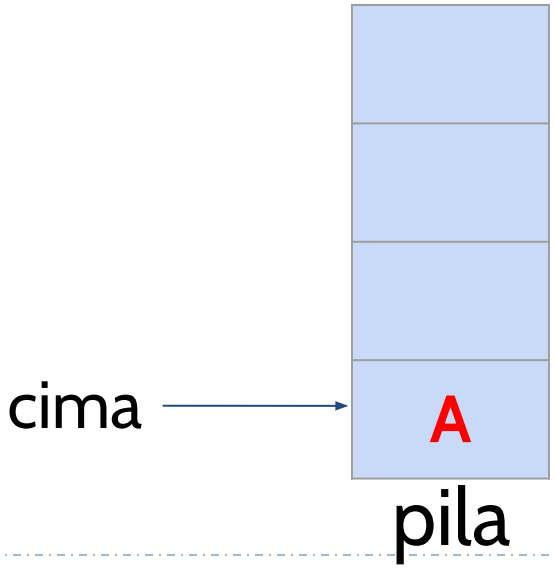
- ¿Cómo **invertir una cadena de texto**?
  - Una vez apilados todos los carácteres, desapilamos cada carácter de la pila, hasta que quede vacía.
  - Cada carácter desapilado se concatena al final de la cadena que devolveremos como salida.



Output: RO

Resolución de problemas usando pilas

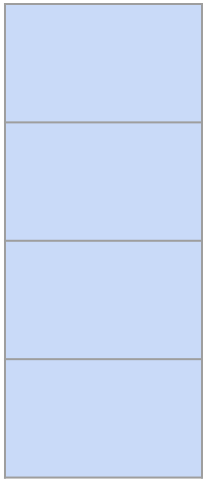
- ¿Cómo **invertir una cadena de texto**?
  - Una vez apilados todos los carácteres, desapilamos cada carácter de la pila, hasta que quede vacía.
  - Cada carácter desapilado se concatena al final de la cadena que devolveremos como salida.



Output: ROM

Resolución de problemas usando pilas

- ¿Cómo **invertir una cadena de texto**?
  - Una vez apilados todos los carácteres, desapilamos cada carácter de la pila, hasta que quede vacía.
  - Cada carácter desapilado se concatena al final de la cadena que devolveremos como salida.



pila

Output: ROM**A**

Resolución de problemas usando pilas

- Función que recibe una cadena de texto y devuelve su inversa

Resolución de problemas usando pilas

- La evaluación de una expresión aritmética (o lógica), implica comprobar si sus **paréntesis están correctamente balanceados**:

Expresión	**Paréntesis balanceados?**
x= (((y+2)*5)/2 - 5) * 10	✓
(()())	✓
((()()))	✗

Resolución de problemas usando pilas

- En una expresión aritmética o lógica, los paréntesis están balanceados si:
  - Cada símbolo de apertura tiene su correspondiente símbolo de cierre.
  - Los pares de paréntesis están correctamente anidados.

Resolución de problemas usando pilas

Algoritmo para comprobar si los paréntesis están balanceados:

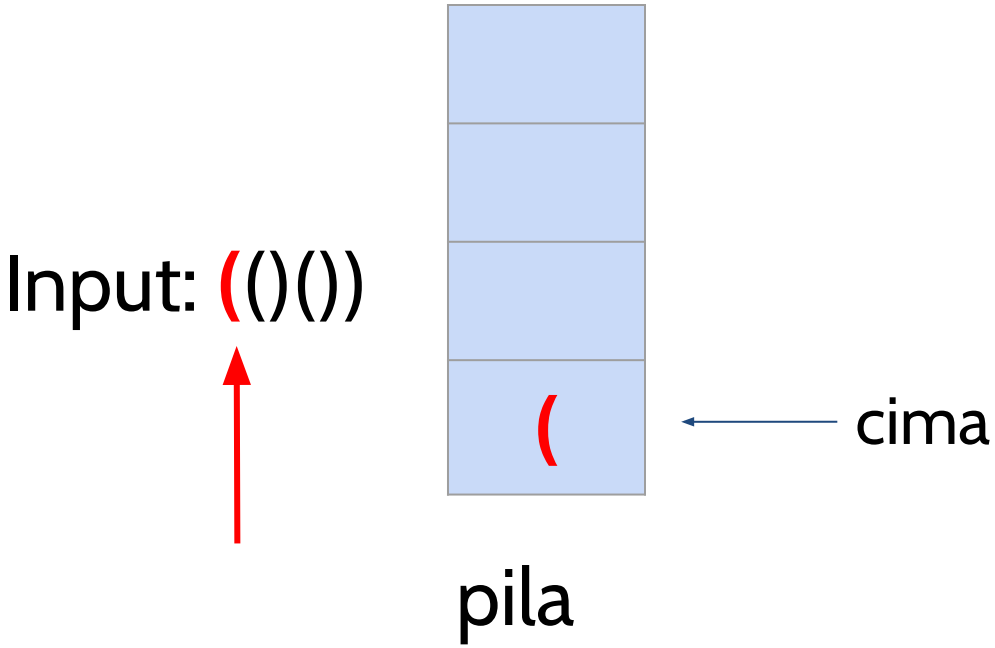
- Crear una pila vacía, que se utilizará únicamente para almacenar los paréntesis de apertura.

Resolución de problemas usando pilas

Algoritmo para comprobar si los paréntesis están balanceados:

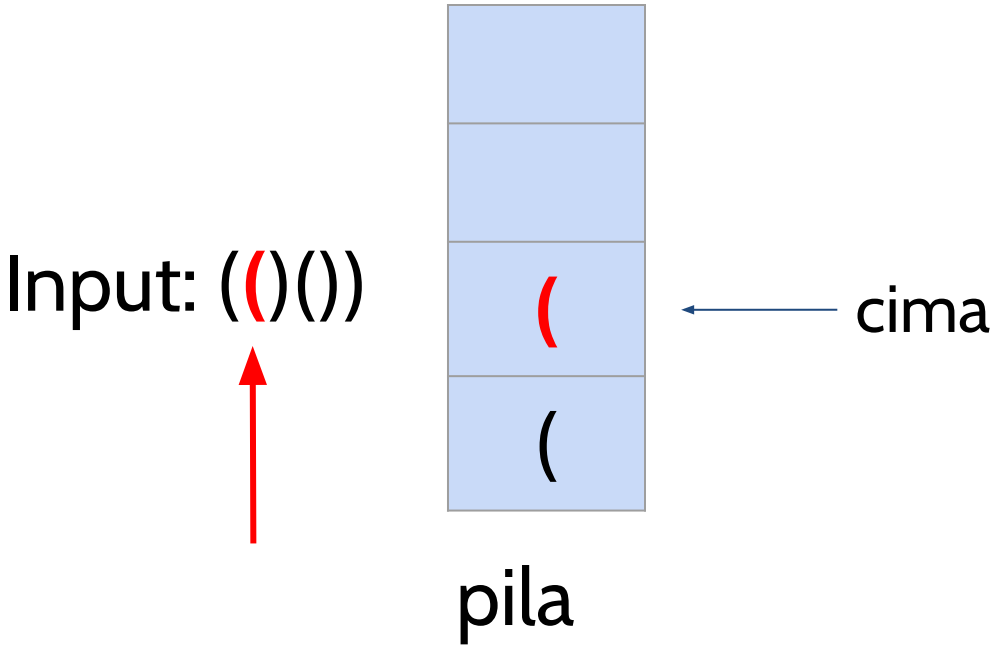
- Crear una pila vacía, que se utilizará únicamente para almacenar los paréntesis de apertura.
- **Leer cada paréntesis** de la expresión, de izquierda a derecha:
  - Si el paréntesis es de apertura, '**(**', lo **apilamos**.
  - Si el paréntesis es de cierre, ')', **desapilamos**.

Resolución de problemas usando pilas



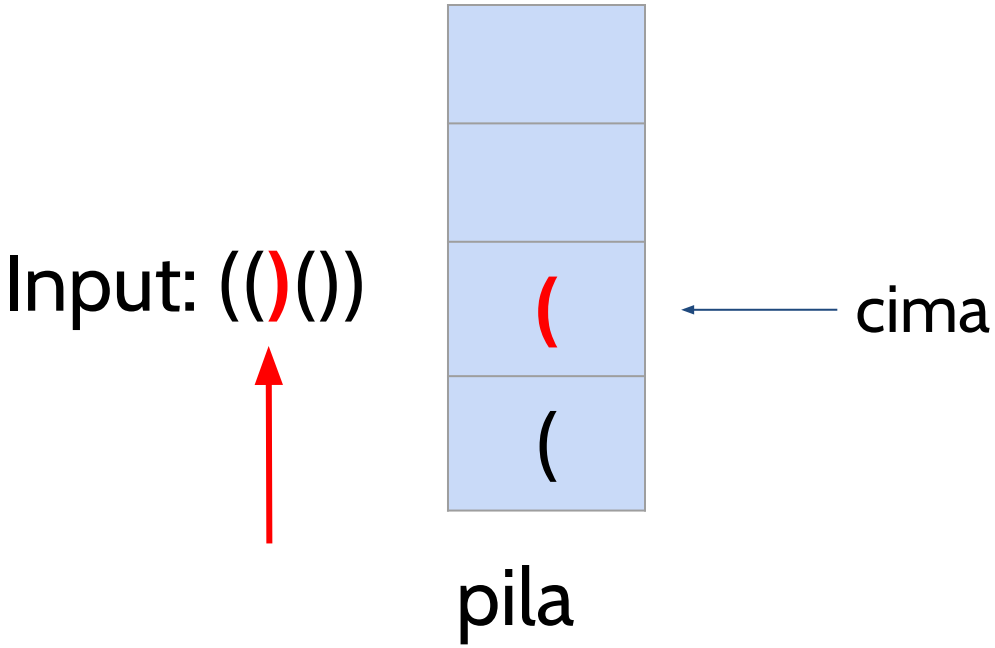
Es un paréntesis de apertura, debemos apilar.

Resolución de problemas usando pilas



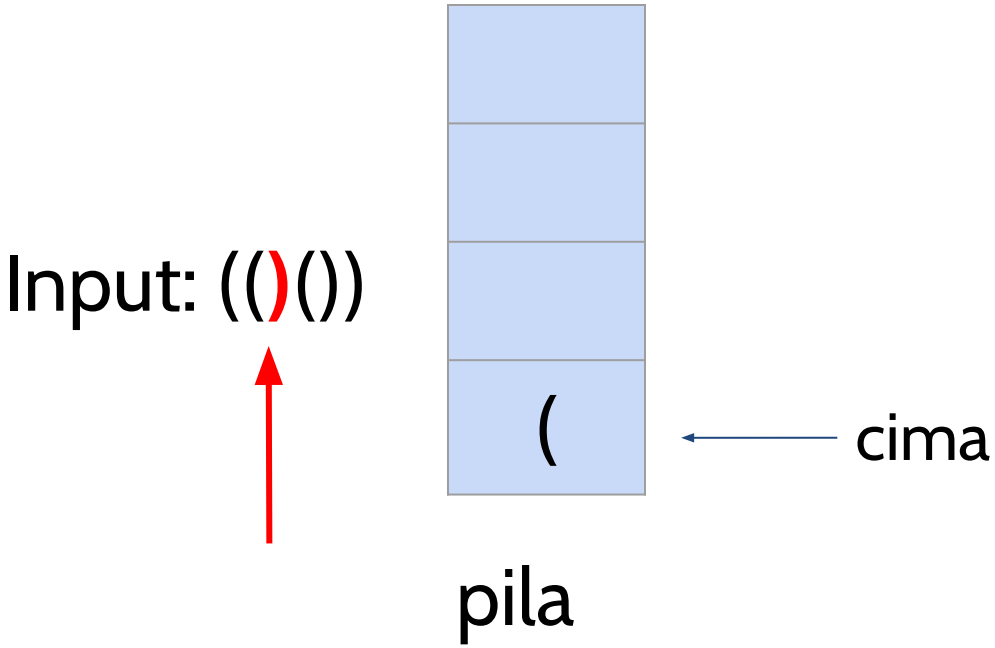
Es un paréntesis de apertura, debemos apilar

Resolución de problemas usando pilas



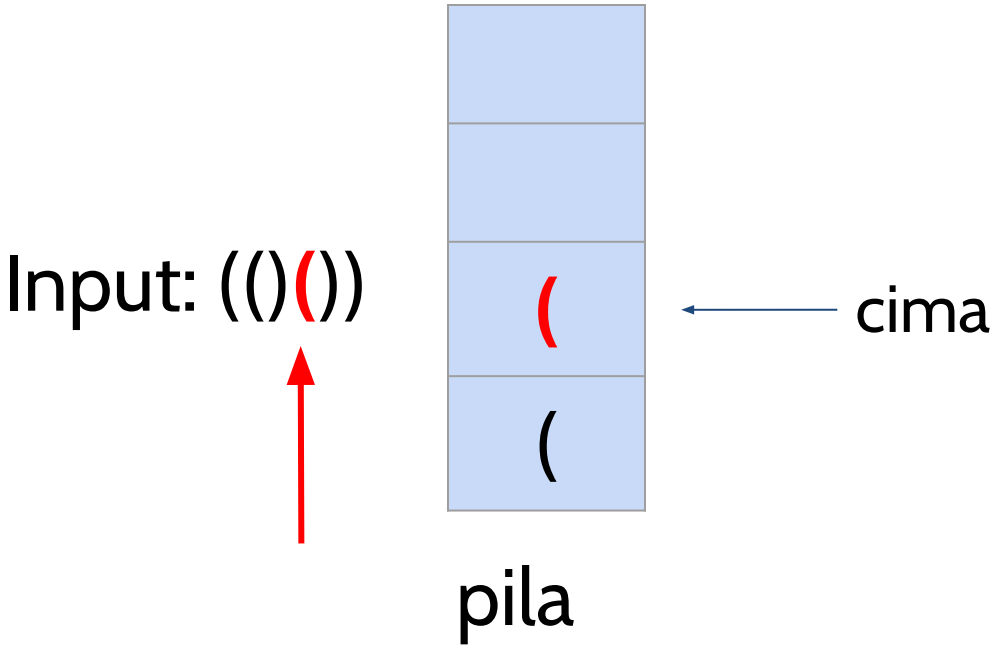
Es un paréntesis de cierre, debemos desapilar

Resolución de problemas usando pilas



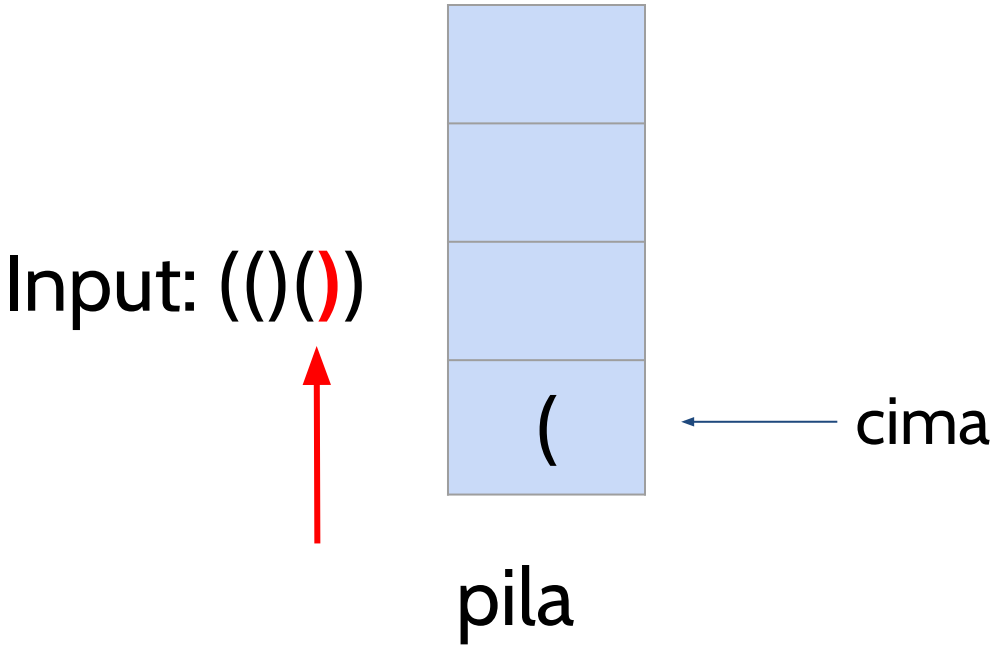
Después de desapilar, seguimos teniendo en la pila el primer paréntesis que apilamos.

Resolución de problemas usando pilas



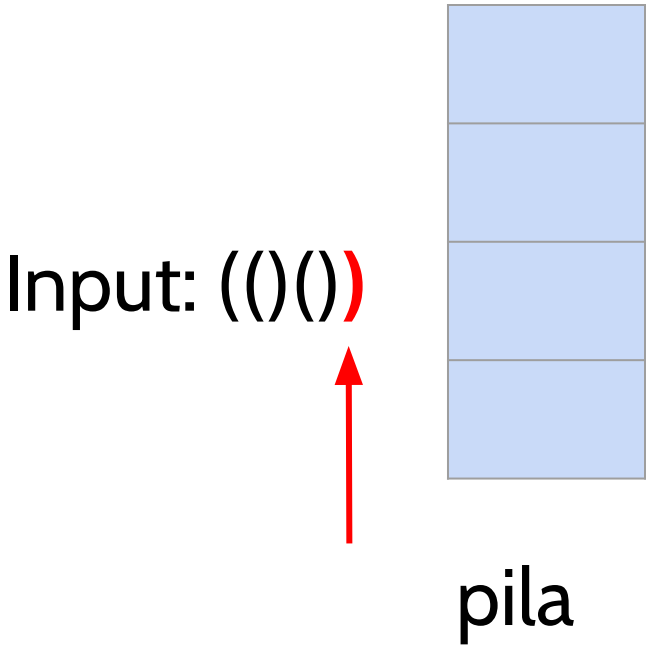
Es un paréntesis de apertura, debemos apilar

Resolución de problemas usando pilas



Después de desapilar, seguimos teniendo en la pila el primer paréntesis que apilamos.

Resolución de problemas usando pilas



Hemos terminado de leer todos los paréntesis

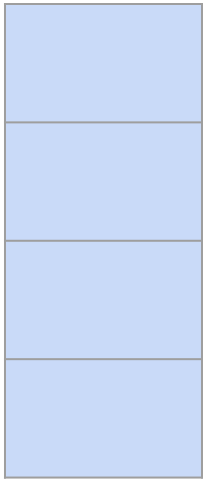
Resolución de problemas usando pilas

Algoritmo para comprobar si los paréntesis están balanceados:

- Cuando hemos leído todos los paréntesis de la expresión, **comprobamos si la pila está vacía.**
  - **Si está vacía,** significa que los paréntesis están balanceados y devolveremos **cierto**.
  - **Si no está vacía**, significa que los paréntesis no están balanceados y devolveremos **falso**.

Resolución de problemas usando pilas

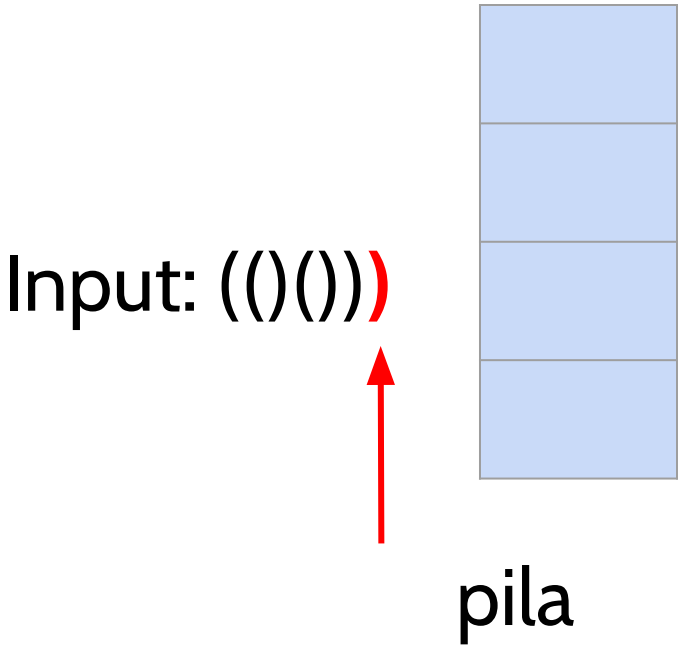
Input: (()())



pila

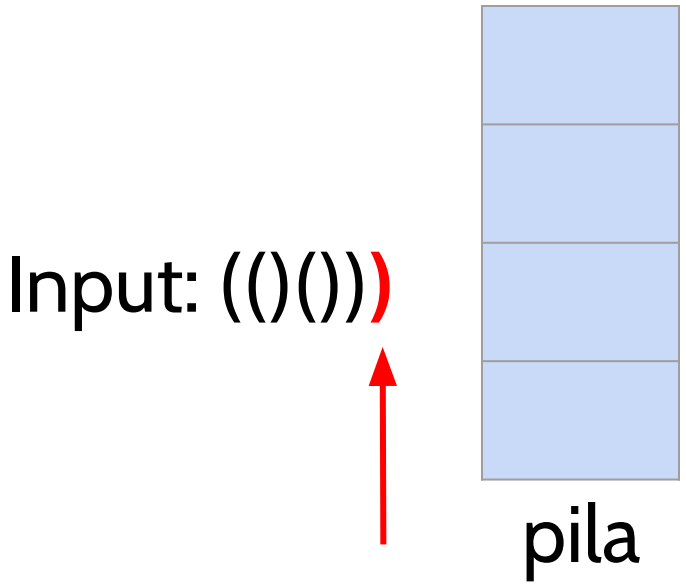
En nuestro ejemplo, devolveremos cierto, porque la pila está vacía.

Resolución de problemas usando pilas



Si la expresión de entrada tuviera un paréntesis de cierre más al final, ¿qué ocurriría?

Resolución de problemas usando pilas



- Como el paréntesis es de cierre, tenemos que desapilar, pero la pila está vacía!!!
- Esto significa que para ese paréntesis de cierre, no hay un paréntesis de apertura. Por tanto, los paréntesis en la expresión '(()()))' no estarían balanceados. Devolvemos falso.

Resolución de problemas usando pilas

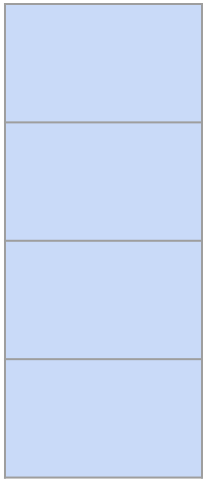
Algoritmo para comprobar si los paréntesis están balanceados:

- Cuando hemos leído todos los paréntesis de la expresión, **comprobamos si la pila está vacía.**
  - **Si está vacía,** significa que los paréntesis están balanceados y devolveremos **cierto**.
  - **Si no está vacía**, significa que los paréntesis no están balanceados y devolveremos **falso**.

Resolución de problemas usando pilas

¿Este algoritmo es correcto y robusto?, es decir, ¿es capaz de resolver todos los casos?. Veamos el siguiente ejemplo:

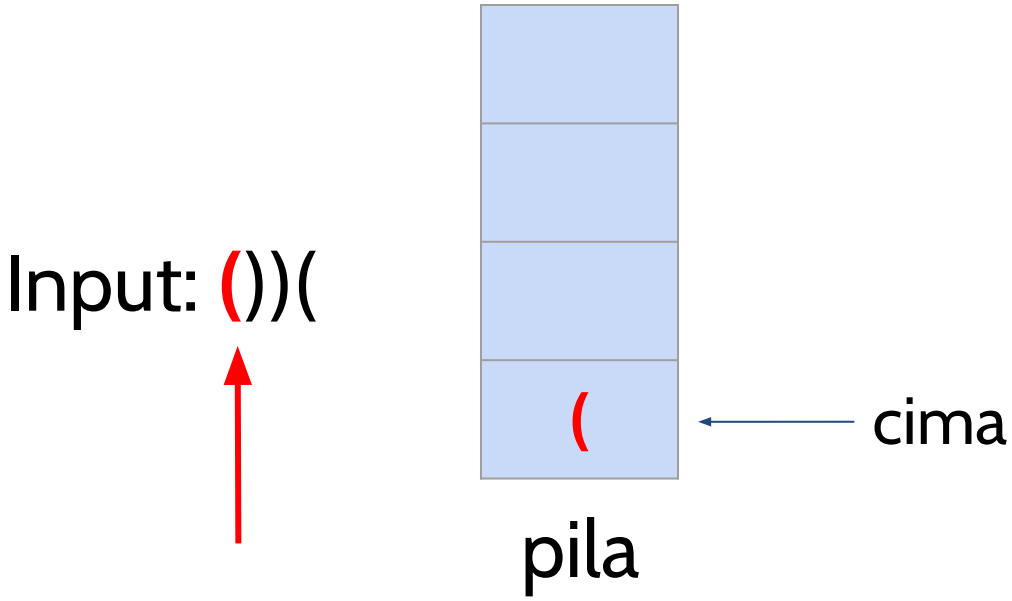
Input: ())(



pila

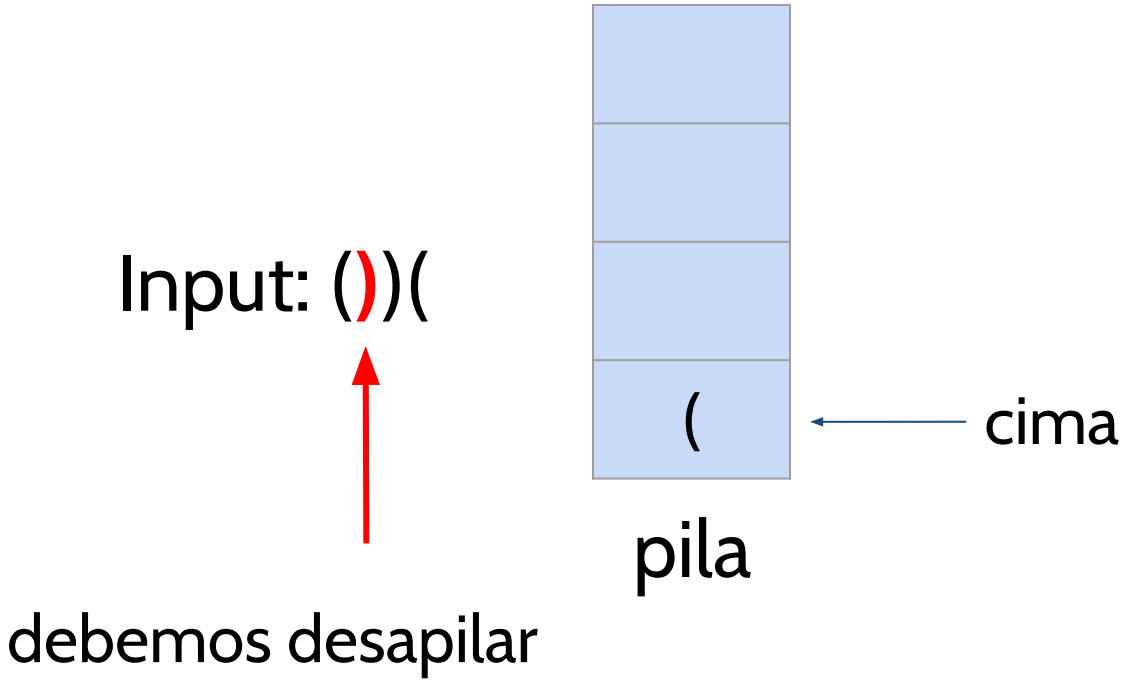
Resolución de problemas usando pilas

¿Este algoritmo es correcto y robusto?, es decir, ¿es capaz de resolver todos los casos?. Veamos el siguiente ejemplo:



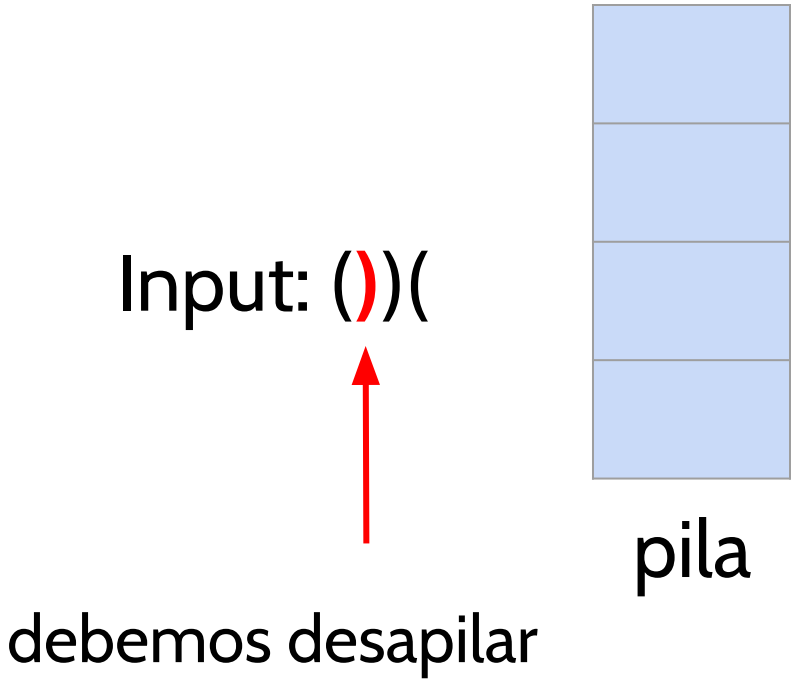
Resolución de problemas usando pilas

¿Este algoritmo es correcto y robusto?, es decir, ¿es capaz de resolver todos los casos?. Veamos el siguiente ejemplo:

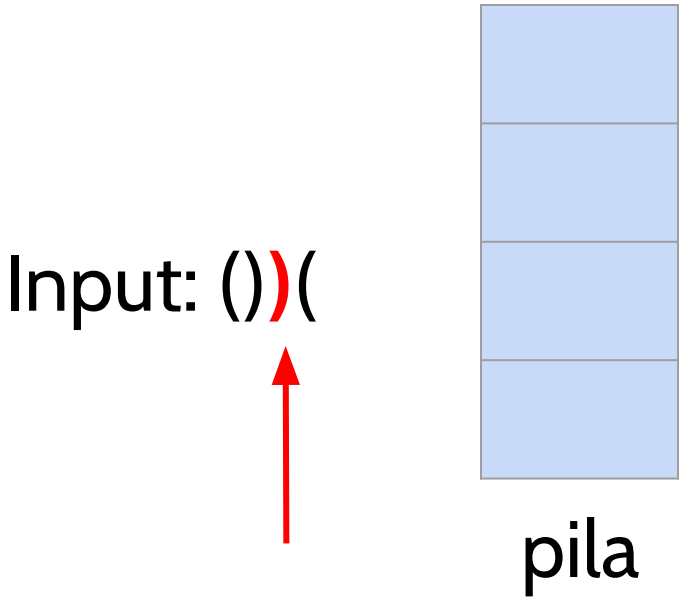


Resolución de problemas usando pilas

¿Este algoritmo es correcto y robusto?, es decir, ¿es capaz de resolver todos los casos?. Veamos el siguiente ejemplo:



Resolución de problemas usando pilas



- Debemos desapilar de nuevo, pero la pila está vacía. Eso significa que en la pila no hay un paréntesis de apertura para el paréntesis de cierre que estamos procesando.
- Por tanto, la expresión '())(', no está balanceada. Tenemos que devolver 'Falso', aunque aún queden paréntesis por leer.

Resolución de problemas usando pilas

- Función que recibe una secuencia de paréntesis y comprueba si están bien balanceados.

Resolución de problemas usando pilas

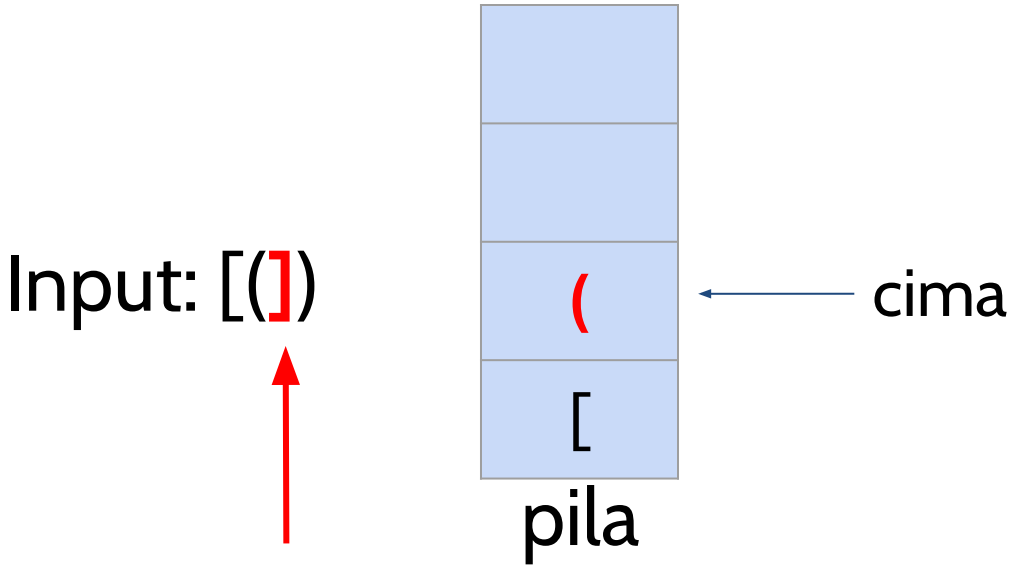
Ejercicio: Considera que las expresiones pueden contener distintos tipos de símbolos: '(', ')', '[', ']', '{', '}'.

- '[(){{}}()]' está balanceada.
- '[()]' está balanceada.
- '[(])' no está balanceada.

El algoritmo es similar al anterior, pero además para comprobar que los paréntesis estén correctamente anidados, tendrás que comprobar que el paréntesis de cierre que lees y el paréntesis de apertura que sacas de la pila, son del mismo tipo.

Resolución de problemas usando pilas

En el siguiente ejemplo, vemos que el paréntesis de cierre que estamos leyendo, ‘]’, no es del mismo tipo que el paréntesis de apertura, ‘(‘, que hay en la cima de la pila. Por tanto, ya podemos devolver ‘False’.



Resolución de problemas usando pilas

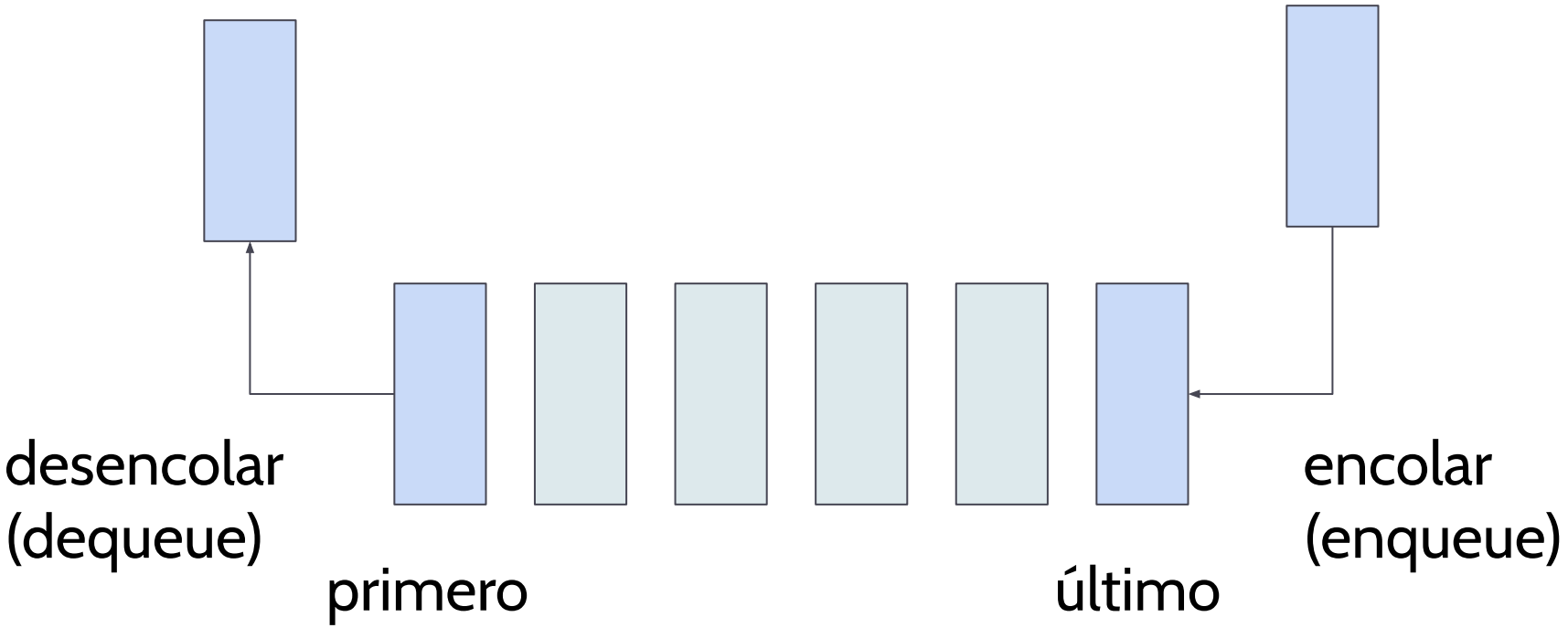
- Función que recibe una secuencia de símbolos de apertura y cierre y comprueba si están bien balanceados.

Índice

- TAD Pila
- **TAD Cola**
- TAD Lista
  - Implementación basada en lista simplemente enlazada
  - Implementación basada en lista doblemente enlazada.

TAD Cola (queue)

- TAD lineal basada en el principio **FIFO (first-in, first-out)**.



TAD Cola (queue)

TAD lineal que permite almacenar y recuperar datos siguiendo el modo de acceso FIFO (first in, first out).

Operaciones:

- **Cola()** (__init__ en Python)**:** crea una cola vacía.
- **encolar**(e): añade el elemento e al final de la cola. No devuelve nada.
- **desencolar**(): borra y devuelve el primer elemento de la cola. La cola es modificada. Si la cola está vacía, muestra un mensaje de error y devuelve un objeto nulo (None).

TAD Cola (queue)

- **primero**(): devuelve el primer elemento de la cola, sin eliminarlo. Es decir, la cola no es modificada. Si la cola está vacía, muestra un mensaje de error y devuelve un objeto nulo (None).
- **tamaño**() (__len__ en Python): devuelve el número de elementos de la cola.
- **está_vacía**(): devuelve cierto si la cola está vacía, y falso en otro caso.

Implementación TAD Cola

- Un array (lista de Python) nos permite fácilmente implementar el TAD Cola.
- Los elementos se van almacenando por orden de llegada.
- Únicamente permitiremos las operaciones definidas en el TAD Cola, en concreto:
  - encolar: que será añadir un elemento al final de la lista que implementa la cola.
  - desencolar: que será eliminar el primer elemento de la lista que implementa la cola.

Implementación TAD Cola (queue)

Implementación del TAD cola basada en array (lista de Python)

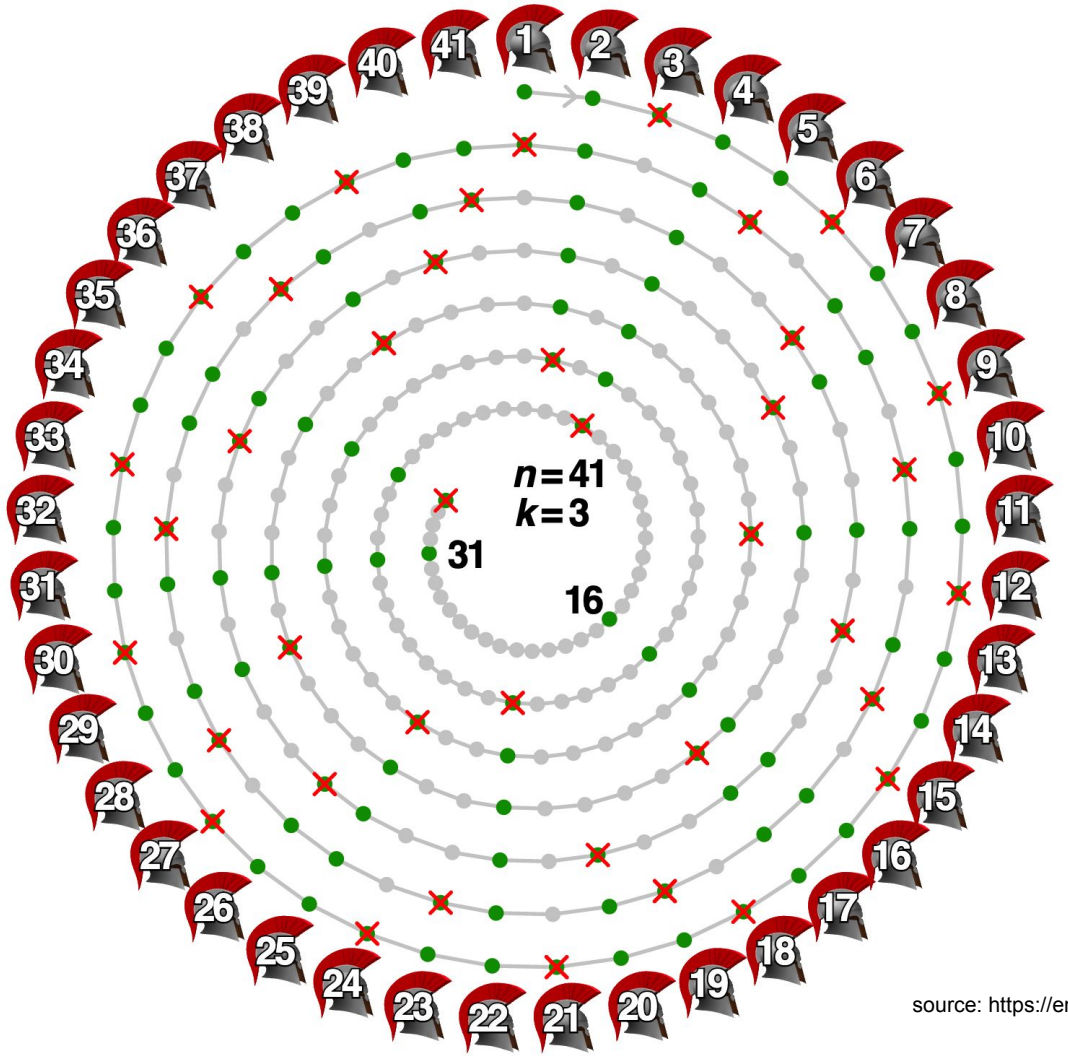
Problema de Flavius Josephus

- Es un problema muy popular en Matemáticas y Computación.
- Josephus era un militar historiador judío del siglo I dc. Él y sus soldados (en total 40), quedaron atrapados en una cueva y rodeados de romanos.
- Para evitar ser capturados, decidieron suicidarse y quitarse la vida entre ellos.
- Josephus (que no quería morir), propuso el siguiente método:

Problema de Flavius Josephus

- Formarían un círculo donde cada soldado es identificado con un número del 1 al 41.
- Contarían de tres en tres, ejecutando al soldado en cada tercera posición.
- El proceso continúa hasta que únicamente queda un soldado, que debería suicidarse.
- Josephus no quería morir así que fue capaz de calcular la posición del último superviviente.
- Existen distintas variantes del problema. En una de ellas, el algoritmo termina cuando aún quedan dos supervivientes que escaparon juntos.
- Existen distintas soluciones para el problema (basada en arrays circulares). Hoy estudiaremos la solución basada en colas.

Problema de Flavius Josephus



source: https://en.wikipedia.org/wiki/File:Josephus_problem_41_3.svg

Implementación problema de Josephus

- Desarrolla una función en Python, josephus, que reciba dos argumentos, n y k:
  - n es el número de soldados,
  - k el contador para cada paso. Es decir, saltaremos a k-1 soldados, y el k-ésimo soldado será ejecutado.
- La función debe devolver la posición del superviviente para n y k.

Solución

Resolución de problemas basados en Cola

- Desarrolla una clase en Python, ***PrinterQueue***, que permita gestionar los trabajos que se mandan a una impresora de red.
- Cada trabajo es identificado por el nombre del fichero a imprimir y el usuario que solicita la impresión.
- Los trabajos deben ser impresos por orden de llegada.
- Para simular la impresión de un trabajo simplemente vamos a mostrar un mensaje con los datos del trabajo.

Solución

Implementar TAD Cola doble (dqueue)

- Una cola doble es un TAD que permite generalizar al TAD cola. En este nuevo TAD, los elementos pueden ser añadidos y borrados tanto del principio como del final de la cola doble.
- Las operaciones que se van a permitir son:
  - Crear una cola doble vacía.
  - add_first(e): que recibe un elemento y lo añade al principio de la cola doble.
  - add_last(e): que recibe un elemento y lo añade al final de la cola doble.

Implementar TAD Cola doble (dqueue)

- Más operaciones:
  - remove_first(): que borra y devuelve el primer elemento que hay en la cola doble. Si la cola está vacía, mostrará un mensaje de error y devolverá el objeto nulo (None).
  - remove_last(): que borra y devuelve el último elemento que hay en la cola doble. Si la cola está vacía, mostrará un mensaje de error y devolverá el objeto nulo (None).

Implementar TAD Cola doble (dqueue)

- Más operaciones:
  - first(): devuelve el primer elemento de la cola doble. La cola no es modificada. Si la cola está vacía, devuelve el objeto nulo (None).
  - last(): devuelve el último elemento de la cola doble. La cola no es modificada. Si la cola está vacía, devuelve el objeto nulo (None).
  - tamaño(): devuelve el número de elementos en la cola doble.
  - está vacía(): devuelve cierto si la cola está vacía y falso en otro caso.

Solución

Resumen

- Una pila es un TAD lineal cuyas operaciones apilar y desapilar, siguen el principio LIFO (last in, first out).
- Una cola es un TAD lineal cuyas operaciones encolar y desencolar, siguen el principio LIFO (first in, first out).
- Hemos proporcionado implementaciones para ambos TAD basadas en arrays (listas de Python).
- Ambas estructuras lineales pueden ser utilizadas en la resolución de problemas típicos de Computación.