Pilas & Colas en Python

Antonio Espín Herranz

Introducción

- Para la implementación de Pilas y Colas en Python como primera opción podemos utilizar el tipo **List** (utilizando métodos **append** y **pop**)

- Se pueden encapsular una lista dentro de una **clase** y dotarla de métodos como pueden ser apilar, desapilar (en el caso de la Pila) y en el caso de la cola encolar, desencolar.

Pila: implementación con list

- letters = []

- # Let's push some letters into our list
- letters.append('c')
- letters.append('a')
- letters.append('t')
- letters.append('g')

- # Now let's pop letters, we should get 'g'
- last_item = letters.pop()
- print(last_item)

- # If we pop again we'll get 't'
- last_item = letters.pop() → Elimina el último elemento de la lista (si no se indica el index). Tratamiento LIFO
- print(last_item)

- # 'c' and 'a' remain
- print(letters) # ['c', 'a']

Cola: implementación con list

- fruits = []

- # Let's enqueue some fruits into our list
- fruits.append('banana')
- fruits.append('grapes')
- fruits.append('mango')
- fruits.append('orange')

- # Now let's dequeue our fruits, we should get 'banana'
- first_item = fruits.pop(0)
- print(first_item)

- # If we dequeue again we'll get 'grapes'
- first_item = fruits.pop(0) # Eliminamos el primero, tratamiento FIFO
- print(first_item)

- # 'mango' and 'orange' remain
- print(fruits) # ['c', 'a']

Clase deque

- Además disponemos de la clase **deque**() dentro del módulo **collections** de la librería.
  - Con los **métodos**:
    - **append**: para añadir al final.
    - **pop** quitar el último.
    - **popLeft** quitar el primero.

```
from collections import deque

# you can initialize a deque with a list
numbers = deque()

# Use append like before to add elements
numbers.append(99)
numbers.append(15)
numbers.append(82)
numbers.append(50)
numbers.append(47)

# You can pop like a stack
last_item = numbers.pop()
print(last_item) # 47
print(numbers) # deque([99, 15, 82, 50])

# You can dequeue like a queue
first_item = numbers.popleft()
print(first_item) # 99
print(numbers) # deque([15, 82, 50])
```

Implementaciones con clases

```
# A simple class stack that only allows pop and push operations
class Stack:

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

    def pop(self):
        if len(self.stack) < 1:
            return None
        return self.stack.pop()

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

    def size(self):
        return len(self.stack)
```

```
# And a queue that only has enqueue and dequeue operations
class Queue:

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

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

    def dequeue(self):
        if len(self.queue) < 1:
            return None
        return self.queue.pop(0)

    def size(self):
        return len(self.queue)
```