

Pilas y colas en Python

Diego Alexander Murillo Suárez

Fundación universitaria del areandina

Modelos de programación II - IS - 202460-6A - 61

Deyvis Morales

Sábado 10 de agosto del 2024

Introducción

En el siguiente documento se presentará el entendimiento de qué es una Pila y qué es una cola con el lenguaje de programación Python. Desde la implementación hasta el por qué son estructuras fundamentales para el mundo actual, todo esto utilizando librerías que se pueden encontrar en el mundo profesional, usando la responsabilidad única (la letra “S” dentro de los principios solid) y el lenguaje inglés a la hora de escribir código para la mantenibilidad de estos ejemplos que son pequeños y escalables.

Pilas y Colas

Objetivos

Con este documento quiero que entiendan las estructuras llamadas y Pilas y cómo las pueden implementar ambas estructuras en el mundo actual con ayuda de Python, intentando darles diferentes perspectivas de cómo se pueden realizar y cómo poder optimizarlas.


Primer ejercicio

Realice un programa que permita insertar N elementos en una lista tipo pila y colas enlazada simple, posteriormente realice:

- Mostrar los datos de la lista.
- La cantidad de números pares que hay en la lista
- El promedio de la lista.
- Último dato de la lista

Solución

Para comenzar, declaramos una variable global de tipo lista la cual tendrá una estructura de tipo pila, que en principio estará vacía y luego la llenaremos por medio de una función con una cantidad N de elementos enteros y aleatorios gracias a la librería random.



```
1 import random
2 data_list = []
3 def insert_random_data_into_list(quantity):
4     count = 0
5     while count < quantity:
6         value = random.randint(1,quantity)
7         data_list.append(value)
8         count += 1
```

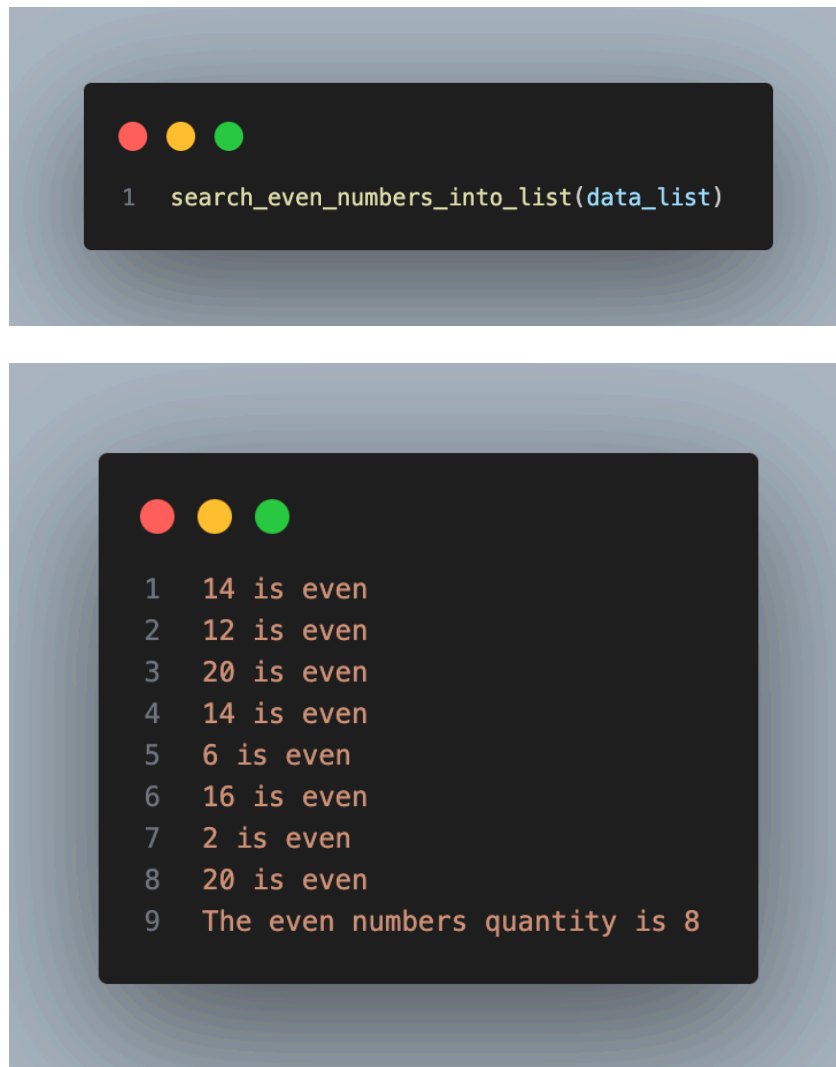
Luego de esto, ejecutaremos la función e imprimimos la lista con los datos ya llenos.

```
1 insert_random_data_into_list(20)
2 print(data_list)
```

```
1 [11, 5, 15, 14, 9, 1,
2  1, 12, 9, 7, 13, 7, 20, 14, 17, 1, 6, 16, 2, 20]
```

Ahora debemos revisar los elementos que sean pares dentro de esta respuesta, para eso crearemos una función con ese objetivo e imprimiremos cada número par y el total de números pares dentro de la lista.

```
1 def average_data_list_without_functions(data_list):
2     count = 0
3     addition = 0
4     data_quantity = len(data_list)
5     while count < data_quantity:
6         addition += data_list[count]
7         count += 1
8     average_data = addition / data_quantity
9     print(f"The average data is {average_data}")
```



```
1 search_even_numbers_into_list(data_list)
```

```
1 14 is even
2 12 is even
3 20 is even
4 14 is even
5 6 is even
6 16 is even
7 2 is even
8 20 is even
9 The even numbers quantity is 8
```


Ahora debemos sacar el promedio de la lista, para esto se hizo de dos maneras distintas, en la primera no usamos ninguna librería de CPython, en cambio la segunda función sí se utilizan estas funciones. Después de este ejercicio explicaremos un poco más a fondo cuál es la más óptima. Ambas tienen la misma salida.

```
1 def average_data_list_without_functions(data_list):  
2     count = 0  
3     addition = 0  
4     data_quantity = len(data_list)  
5     while count < data_quantity:  
6         addition += data_list[count]  
7         count += 1  
8     average_data = addition / data_quantity  
9     print(f"The average data is {average_data}")  
10  
11 def average_data_list_with_functions(data_list):  
12     print(f"The average data is {sum(data_list)/len(data_list)} ")
```


```
1 # The average data is 10.0  
2 # The average data is 10.0
```

Si ustedes ven, la primera función es más flexible, mientras que la segunda es más simple, a primera vista se podría decir que la primera función es más “rápida” debido a que estamos haciendo el proceso más manual, sin embargo la segunda función tiene las librerías de CPython, las cuales ayudan a que Python ejecute estas operaciones en menos tiempo y consumiendo menos energía al ser funciones creadas en C.

Finalmente debemos ver cuál es el último número de la lista o cola que tenemos. Acá también disponemos de 2 ejemplos. Al final de estos 2 ejercicios hablaremos de las 2 diferencias y cuál es el más óptimo.



```
1 def last_number_in_data_list_with_negative_call(data_list):
2     print(f"The last number in the list is {data_list[-1]}")
3
4
5 def last_number_in_data_list_without_negative_call(data_list):
6     data_quantity = len(data_list) - 1
7     last_number = data_list[data_quantity]
8     print(f"The last number in the index is {last_number}")
9
```



```
1 # The last number in the list is 20
2 # The last number in the index is 20
```

La diferencia entre estos es la notación, para aquellos desarrolladores en Python hay una manera simple y concisa de averiguar el último número de la lista, y esto es realizando lo que hace la primera función, consumiendo menos recursos y haciendo más entendible el código.

Segundo ejercicio

Crea una lista enlazada simple de tipo Pila y Cola que permita almacenar los n datos de una persona como son: código, nombre, teléfono y edad.

- Mostrar lista de los elementos.
- Eliminar el primer elemento de la lista y mostrar nuevamente la lista.
- Contar los elementos que quedan en la lista.

Solución

Si vemos en el primer ejercicio, nos enfocamos más en la estructura de datos de tipo Pila, el segundo está más enfocado a usar la estructura de datos de tipo Cola. De igual manera, al tener que insertar más datos reales, he decidido utilizar las librerías de faker y random en el código para las soluciones.

Comencemos creando una función que me llene la lista como si esta fuera un objeto y dentro de este se puedan crear una lista con los diferentes datos de las personas, la llamaremos e imprimimos los valores que hay dentro de la lista.

```
1 person_data = []
2 fake = Faker()
3
4 def insert_random_data_into_list(quantity:int):
5     count = 0
6     while count < quantity:
7         data = [
8             fake.unique.random_int(min=1,max=quantity), # code
9             fake.name(), # name
10            fake.phone_number(), # phone number
11            random.randint(0,100) # age
12        ]
13        person_data.append(data)
14        count += 1
```



```
1 insert_random_data_into_list(5)
2 print(f"Full queue: {person_data}")
3 """"
4 Output:
5 Full queue:
6 [
7  [4, 'Bradley Duncan', '+1-710-232-7477', 16],
8  [2, 'Dennis Robinson', '970-775-9800', 94],
9  [3, 'Mandy Munoz', '327-543-4639x210', 11],
10 [1, 'Stephanie Powell', '+1-324-743-6911x4083', 43],
11 [5, 'Shaun Garcia', '(950)748-3042', 96]
12 ]
13 """"
```

Luego de esto debemos eliminar el primer elemento de la lista, el elemento `.pop()` también recibe un parámetro para saber cuál índice debe eliminar, en este caso, necesitamos eliminar la posición 0. Para esto crearemos una función que lo haga y luego de eso imprimimos la lista con el primer objeto en la lista ya eliminado.

```
1 def remove_first_item_in_list(data_list:list):
2     data_list.pop(0)
```

```
1 remove_first_item_in_list(person_data)
2 print(f" Queue without first element: {person_data}")
3 """
4 Output:
5 Queue without first element: [
6 [2, 'Dennis Robinson', '970-775-9800', 94],
7 [3, 'Mandy Munoz', '327-543-4639x210', 11],
8 [1, 'Stephanie Powell', '+1-324-743-6911x4083', 43],
9 [5, 'Shaun Garcia', '(950)748-3042', 96]
10 ]
11 """
```

Como podemos ver, el registro que estaba en la posición 0 no aparece, esto debido a que se eliminó. Ahora imprimimos la cantidad de datos que hay en la lista, para esto usaremos la función de CPython llamada `len()`.

```
1 print(len(person_data))
2 # Output: 4
```

Discusión y conclusiones

Python tiene una simpleza increíble para estas 2 estructuras de datos que se pueden comprender de una manera más simple en comparación con otros lenguajes de programación. Gracias a todo esto, Python es uno de los lenguajes más utilizados al día de hoy para la arquitectura de eventos, la cual tiene mucha relación con lo que son pilas y colas, evitando que hacer 50.000 peticiones a tu servidor no lo haga caer, todo esto de una manera asíncrona. A pesar de todas estas cosas positivas Python consume demasiada energía, por lo cuál tienen que tener diferentes intérpretes para realizar ciertas operaciones sin consumir tantos recursos, CPython es un intérprete mencionado en este documento, a pesar de eso, hay otros adicionales como lo son, Jython y IronPython. En las referencias dejaré datos, documentación y el repositorio en el cual creé estos ejemplos acerca de todo lo mencionado en esta discusión y conclusión.

Referencias

Python Organization. *Random - Generate pseudo-random numbers*. Última actualización de la documentación realizada el 31 de Julio del 2024 de

<https://docs.python.org/es/3/library/random.html>.

Faker documentation. *Welcome to Faker 's documentation!*. Documentación mantenida desde el año 2014 hasta el 2024 de <https://faker.readthedocs.io/en/master/>

Energy Efficiency across Programming Languages (Table 4). Estudio realizado el 23 y 24 de octubre del 2017 de <https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf>.

CPython repository. Última actualización realizada el 10 de Agosto del 2024 de

<https://github.com/python/cpython>

Jython website. *What is Jython?*. Última actualización entre el 28 y 3 de agosto del 2024

<https://www.jython.org/>

Husban. *Difference Between Stack and Queue Data Structures*. Última actualización realizada el 14 de Mayo del 2024 de

<https://www.geeksforgeeks.org/difference-between-stack-and-queue-data-structures/>

Diego M. *Queues and Stacks in Python*. Última actualización el 10 de agosto del 2024 de

<https://github.com/DiegoMurillo7536/Queues-and-Stacks-in-Python>