Programación Imperativa III

Arreglos

- Secuencias de una cantidad fija de valores del mismo tipo.
- Se declaran con un nombre y un tipo:
  - Según el lenguaje, además se debe indicar su tamaño (el cual permanece fijo).
  - Veremos que en Python, los arrays tienen longitud variable.
- Solamente hay valores en las posiciones válidas (dentro de su tamaño).
- Una sóla variable contiene muchos valores:
  - A cada valor se lo accede directamente mediante corchetes.
  - Si *a* es un arreglo de 10 elementos, *a*[5] devuelve el 6to valor.
  - El primer elemento es el de índice 0.

- Supongamos que la variable *a* tiene tipo de dato arreglo de int.
- Podemos ejemplificar que sucede con su referencia en esta simplificación:
  - La variable *a* tiene su referencia en B1.
  - Como es un arreglo de tamaño 4, tiene asociadas 4 posiciones más de memoria.
  - Por ejemplo: *a*[2] tiene el valor de 7 (el valor que está en B3).
  - Si tenemos que describir el estado de la variable *a*, el mismo es [5,6,7,8].
    - *a*[2] no es una variable en sí misma, la variable es *a*.

Memoria

	1	2	3	4	5
A					
B	5	6	7	8	
C					
D					
E					

Variables

Nombre	Tipo	Tamaño	Valor	Referencia
a	Array de Int	4	[5,6,7,8]	B1

**Características **

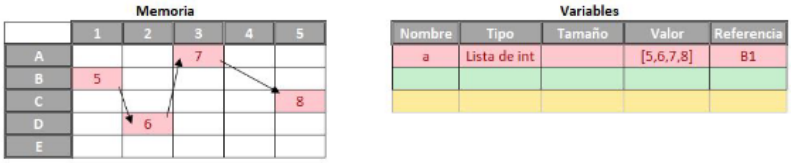
- Estructura de datos lineal: Los arreglos son estructuras de datos lineales que almacenan elementos del mismo tipo en posiciones contiguas de memoria. Estos elementos se pueden acceder y manipular mediante su índice, que indica su posición en el arreglo.

- Tamaño fijo: Los arreglos tienen un tamaño fijo, es decir, se les asigna una cantidad específica de elementos en el momento de su creación y no pueden crecer o reducirse dinámicamente. El tamaño del arreglo se define durante la declaración y no se puede cambiar posteriormente.
- Almacenamiento secuencial: Los elementos en un arreglo se almacenan de manera secuencial en la memoria, lo que permite un acceso eficiente mediante el cálculo del índice base y el desplazamiento. Esto facilita la búsqueda y manipulación de elementos dentro del arreglo.
- Acceso directo a los elementos: Los elementos en un arreglo se pueden acceder directamente mediante su índice. Esto permite un acceso rápido y eficiente a elementos individuales sin la necesidad de recorrer el arreglo secuencialmente.
- Homogeneidad de tipos: Los arreglos en la programación imperativa generalmente contienen elementos del mismo tipo de datos. Todos los elementos almacenados en el arreglo deben ser del mismo tipo, lo que proporciona una estructura homogénea y facilita la manipulación de los datos.
- Eficiencia en operaciones indexadas: Los arreglos son eficientes en operaciones indexadas, como la búsqueda, inserción o eliminación de elementos en una posición específica. Debido a su almacenamiento secuencial y acceso directo a los elementos mediante índices, estas operaciones pueden realizarse en tiempo constante O(1) (en notación de complejidad algorítmica, O(1) significa que el tiempo de ejecución de un algoritmo o una operación es independiente del tamaño del problema o de los datos de entrada. No importa cuántos elementos haya en el arreglo, el tiempo necesario para acceder a un elemento específico es siempre el mismo, ya que se puede calcular su posición directamente).

Listas y Arreglos

Arreglos y Listas

- Ambos representan secuencias de elementos de un tipo.
- Los arreglos suelen tener longitud fija, las listas, no.
- Los elementos de un arreglo pueden accederse en forma independiente y directa:
  - Los de la lista se acceden secuencialmente, empezando por la cabeza,
  - Para acceder al i-ésimo elemento de una lista, hay que obtener i veces la cola y luego la cabeza.
  - Para acceder al i-ésimo elemento de un arreglo, simplemente se usa el índice.



Python

En Python, hay diferencias significativas entre los arrays y las listas. Aquí están las principales diferencias:

- Homogeneidad vs. Heterogeneidad: Los arrays en Python son homogéneos, lo que significa que todos los elementos deben ser del mismo tipo de datos. Por otro lado, las listas en Python son heterogéneas, lo que significa que pueden contener elementos de diferentes tipos de datos.
- Tipo de datos subyacente: Los arrays en Python son implementados mediante el módulo array y utilizan un tipo de datos específico para almacenar los elementos, como enteros de 4 bytes o números de punto flotante de 8 bytes. Las listas en Python son una estructura de datos incorporada y no están restringidas a un tipo de datos específico.
- Flexibilidad en el tamaño: Los arrays en Python tienen un tamaño fijo al ser creados, lo que significa que no se pueden agregar ni eliminar elementos una vez que se ha creado el array. Por otro lado, las listas en Python son estructuras de datos dinámicas y se pueden modificar en tamaño, es decir, se pueden agregar, eliminar o cambiar elementos en cualquier momento.
- Funcionalidad y métodos asociados: Las listas en Python ofrecen una amplia gama de métodos y operaciones integrados que facilitan su manipulación, como agregar elementos al final, insertar elementos en una posición específica, eliminar elementos, buscar elementos, ordenar la lista, entre otros. Los arrays en Python tienen una funcionalidad más limitada en comparación con las listas.
- Eficiencia en el almacenamiento y rendimiento: Los arrays en Python suelen ser más eficientes en términos de almacenamiento y rendimiento en comparación con las listas, especialmente cuando se trata de grandes conjuntos de datos numéricos. Debido a su homogeneidad y uso de tipos de datos específicos, los arrays pueden ocupar menos espacio en memoria y pueden realizar ciertas operaciones más rápidamente que las listas.

Arreglos

- *array* es uno de los módulos que permiten utilizar arreglos (otra posibilidad es NumPy). Al crear el arreglo, se indica su tipo y su contenido inicial.

```
from array import *
   a: array = array(typecode, \[inicializers\])
```

```
from array import *
a: array = array('i', [10, 20, 30])
```

Los tipos de datos disponibles en el módulo array son:

Código de tipo	Tipo C	Tipo Python	Tamaño mínimo en bytes
`'b'`	signed char	int	1
`'B'`	unsigned char	int	1
`'u'`	wchar_t	Carácter unicode	2
`'h'`	signed short	int	2
`'H'`	unsigned short	int	2
`'i'`	signed int	int	2
`'I'`	unsigned int	int	2
`'l'`	signed long	int	4
`'L'`	unsigned long	int	4
`'q'`	signed long long	int	8
`'Q'`	unsigned long long	int	8
`'f'`	float	float	4
`'d'`	double	float	8

Estos tipos de datos se utilizan para crear arrays en el módulo array. Cada tipo de datos se representa mediante una letra que se utiliza como código en los métodos del módulo para especificar el tipo de datos que se desea utilizar al crear un array. Por ejemplo, para crear un array de enteros sin signo de 2 bytes, se utilizaría el tipo de datos "H".

- Operaciones

Sea a un array:

$a[i] \longrightarrow$ obtiene el valor del elemento i de a

$a[i] = x \longrightarrow$ asigna x en el elemento i de a

$a.\text{append}(x) \longrightarrow$ añade x como nuevo elemento de a

$a.\text{remove}(x) \longrightarrow$ elimina el primer elemento en a que coincida con x

$a.\text{index}(x) \longrightarrow$ obtiene la posición donde aparece por primera vez el elemento x

$a.\text{count}(x) \longrightarrow$ devuelve la cantidad de apariciones del elemento x

$a.\text{insert}(p,x) \longrightarrow$ inserta el elemento x delante de la posición p

Listas

- En Python, las listas son un tipo de array.

- En Python, las listas pueden tener elementos de diferentes tipos de datos.
- Al igual que los arrays en Python, tienen tamaño dinámico.

¿Cómo se declaran?

```
variableLista = [] #lista vacia
otraVariableLista = list() #lista vacia
otraVariable = [1, 2, True, 'Hola', 5.8]
unaMas = list([4, 9, False, 'texto'])
```

- **Operaciones**

Sea a una lista:

$a[i] \longrightarrow$ obtiene el valor del elemento i de a

$a[i] = x \longrightarrow$ asigna x en el elemento i de a

$a.\text{append}(x) \longrightarrow$ añade x como nuevo elemento de a

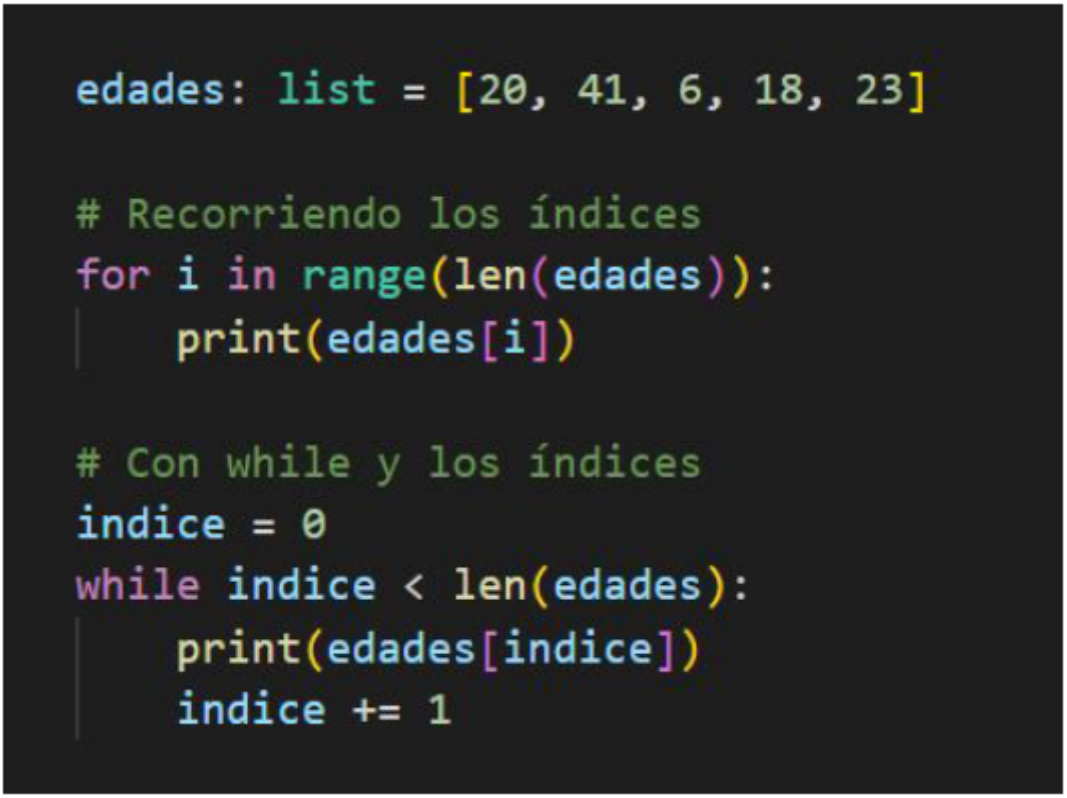
$a.\text{remove}(x) \longrightarrow$ elimina el primer elemento en a que coincida con x

$a.\text{index}(x) \longrightarrow$ obtiene la posición donde aparece por primera vez el elemento x

$a.\text{count}(x) \longrightarrow$ devuelve la cantidad de apariciones del elemento x

$a.\text{insert}(p, x) \longrightarrow$ inserta el elemento x delante de la posición p

- **Recorrer una lista**

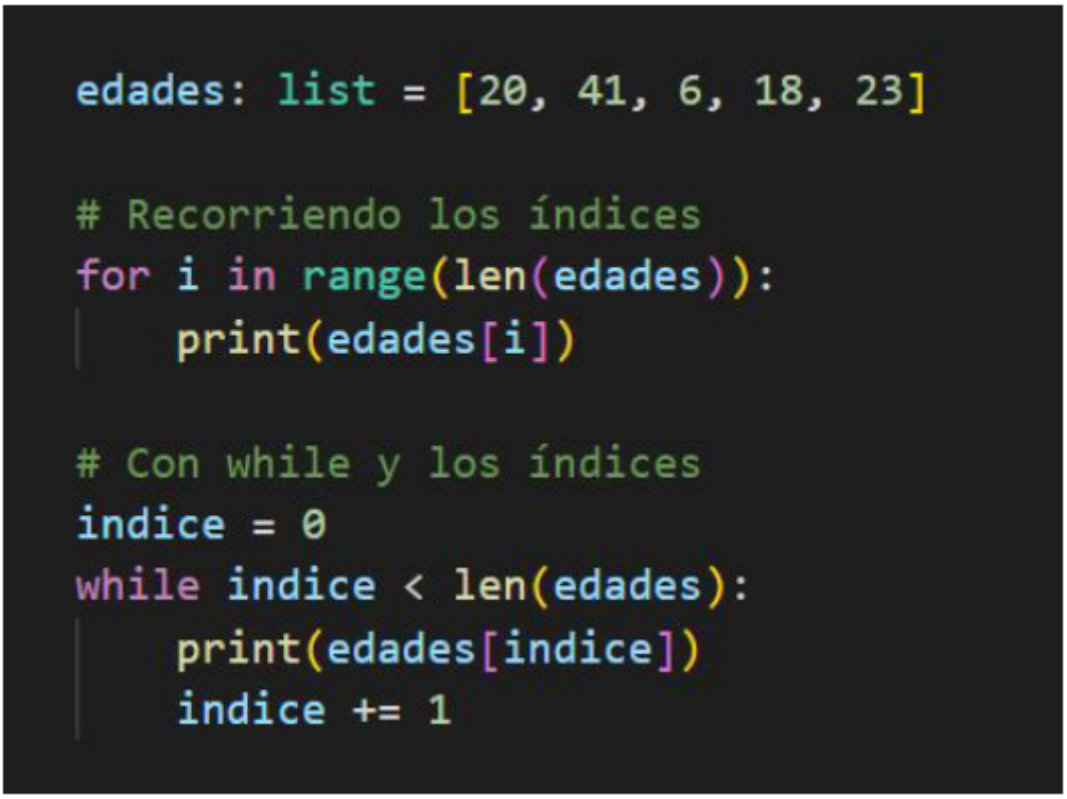


```python
edades: list = [20, 41, 6, 18, 23]

# Recorriendo los índices
for i in range(len(edades)):
    print(edades[i])

# Con while y los índices
indice = 0
while indice < len(edades):
    print(edades[indice])
    indice += 1
```

****



```python
edades: list = [20, 41, 6, 18, 23]

# Recorriendo los índices
for i in range(len(edades)):
    print(edades[i])

# Con while y los índices
indice = 0
while indice < len(edades):
    print(edades[indice])
    indice += 1
```

**

- **Matrices**
  - Podemos pensar una matriz como una lista de listas. Podemos recorrerlas a través de sus índices.

```python
ganadores = [['Messi', 'Cristiano', 'Mbappe'], [7, 5, 1]]

# Recorriendo los índices
# i serían las filas
print("++ Con for - i son filas ++")
for i in range(len(ganadores)):
    for j in range(len(ganadores[i])):
        print("ganadores["+str(i)+"]["+str(j)+"] = " + str(ganadores[i][j]))

print("++ Con while y los índices ++")
# Con while y los índices
fila = 0

while fila < len(ganadores):
    columna = 0
    while columna < len(ganadores[fila]):
        print("ganadores["+str(fila)+"]["+str(columna)+"] = " + str(ganadores[fila][columna]))
        columna += 1
    fila += 1
```

Tipos abstractos de datos (TAD)

Un tipo abstracto de datos (TAD) es un concepto en programación y ciencias de la computación que se refiere a una descripción matemática de una colección de datos y las operaciones que se pueden realizar sobre ellos. Proporciona una **abstracción de alto nivel que oculta los detalles internos de implementación y se centra en la funcionalidad y comportamiento de los datos**. En lugar de enfocarse en cómo se implementan los datos, un TAD define qué operaciones se pueden realizar con esos datos y cómo se comportan esas operaciones. Esto significa que los usuarios pueden utilizar el TAD sin preocuparse por los detalles internos de cómo se almacenan y manipulan los datos.

Un TAD se compone de dos partes principales:

- La especificación de los datos: Describe qué datos están involucrados en el TAD y cómo se organizan. Define las propiedades y restricciones de los datos, así como las operaciones que se pueden realizar sobre ellos.
- La implementación del TAD: Es la forma en que se lleva a cabo la especificación del TAD utilizando estructuras de datos y algoritmos. La implementación puede variar según el lenguaje de programación utilizado y los detalles específicos de la aplicación.

Un Tipo Abstracto de Datos (TAD) es un modelo que define valores y las operaciones que se pueden realizan sobre ellos.

- Se denomina abstracto ya que la intención es que quien lo utiliza, no necesita conocer los detalles de la representación interna o bien el cómo están implementadas sus operaciones.

El tipo lista que estuvimos viendo es un TAD:

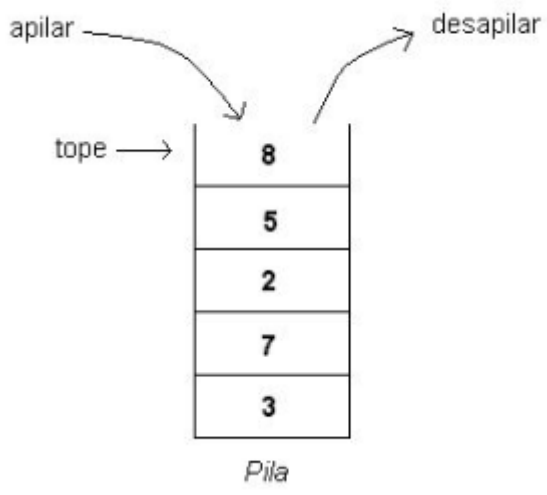
- Se define como una serie de elementos consecutivos
- Tiene diferentes operaciones asociadas: append, remove, etc
- Desconocemos cómo se usa/guarda la información almacenada dentro del tipo

Pila (stack)

Una pila es una estructura de datos lineal que sigue el principio de **Last-In, First-Out (LIFO)**, lo que significa que el último elemento que se inserta en la pila es el primero en ser eliminado. Es similar a una pila de objetos física, donde se pueden agregar elementos en la parte superior y solo se puede acceder al elemento más reciente.

La pila se puede imaginar como una lista vertical de elementos, donde el acceso y la manipulación de los datos se realizan en la parte superior de la pila. Las operaciones principales que se pueden realizar en una pila son:

- Push (apilar): Agrega un elemento en la parte superior de la pila.
- Pop (desapilar): Elimina y devuelve el elemento más reciente de la parte superior de la pila.
- Peek (tope): Obtiene el valor del elemento más reciente sin eliminarlo de la pila.
- isEmpty (vacía): Verifica si la pila está vacía.
- Size (tamaño): Obtiene el número de elementos en la pila.



Las pilas son ampliamente utilizadas en programación para resolver problemas relacionados con el seguimiento de estados, la inversión de secuencias, el manejo de llamadas y retornos de funciones, entre otros. También se utilizan en algoritmos como el recorrido de árboles en profundidad (DFS, por sus siglas en inglés) y la evaluación de expresiones matemáticas.

En Python, el tipo lista provee los métodos necesarios para poder usar una lista como una pila

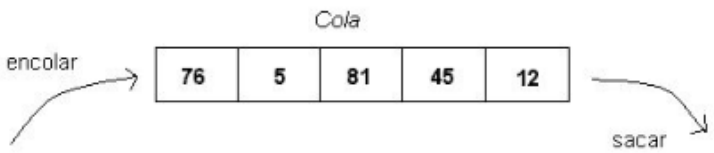
- Operaciones básicas
  - apilar: ingresa un elemento a la pila
    - append
  - desapilar: saca el último elemento insertado
    - pop
  - tope: devuelve (sin sacar) el ultimo elemento insertado
    - a[-1]
  - vacia: retorna verdadero si está vacía
    - len(a)==0

Cola

Una cola es una estructura de datos lineal que sigue el principio de **First-In, First-Out (FIFO)**, lo que significa que el primer elemento que se inserta en la cola es el primero en ser eliminado. Es similar a una cola de objetos física, donde se agregan elementos al final y se eliminan elementos desde el frente.

La cola se puede imaginar como una línea de objetos, donde se agrega un elemento al final de la cola y los elementos se eliminan desde el frente de la cola. Las operaciones principales que se pueden realizar en una cola son:

- Enqueue (encolar): Agrega un elemento al final de la cola.
- Dequeue (desencolar): Elimina y devuelve el elemento del frente de la cola.
- Front (frente): Obtiene el valor del elemento del frente sin eliminarlo de la cola.
- isEmpty (vacía): Verifica si la cola está vacía.
- Size (tamaño): Obtiene el número de elementos en la cola.



Las colas son ampliamente utilizadas en programación para resolver problemas relacionados con la planificación, el procesamiento por lotes, la gestión de tareas en espera y la sincronización de

procesos, entre otros. También se utilizan en algoritmos como el recorrido de árboles en amplitud (BFS, por sus siglas en inglés) y la implementación de buffers.

En Python, el tipo lista provee los métodos necesarios para poder usar una lista como una cola

- Operaciones básicas
  - encolar: ingresa un elemento a la pila
    - append
  - desencolar: saca el primer elemento insertado
    - pop(0)
  - vacia: retorna verdadero si está vacía
    - len(a)==0