

# Introducción a la Programación

## Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2025

Programación Imperativa: Arreglos, Listas, Pilas y Colas

# IP - AED I: Temario de la clase

- ▶ Arreglos, Listas, Pilas y Colas
  - ▶ Arreglos
  - ▶ Arreglos vs Listas
  - ▶ Arreglos en Python
  - ▶ Operaciones básicas de arreglos
  - ▶ Listas en python
  - ▶ Operaciones básicas de listas
  - ▶ ¿Cómo recorrer listas?
  - ▶ Modelado de matrices. ¿Cómo recorrer una matriz?
  - ▶ Iterables y los For
  - ▶ Tipos abstractos de datos
  - ▶ TAD Pila: Definición, operaciones
  - ▶ Implementación en python de una Pila con el tipo de dato List
  - ▶ TAD Cola: Definición, operaciones
  - ▶ Implementación en python de una Cola con el tipo de dato List

# Variables en imperativo

## Repasando

- ▶ Nombre asociado a un espacio de memoria.
- ▶ La variable puede tomar distintos valores a lo largo de la ejecución.
- ▶ En Python se **declaran** dando su nombre (y opcionalmente su tipo)  
x: int # x es una variable de tipo int.  
c: str # c es una variable de tipo string.
- ▶ Programación imperativa:
  - ▶ Conjunto de variables.
  - ▶ Instrucciones que van cambiando sus valores.
  - ▶ Los valores finales, deberían resolver el problema.

# 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 sola 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.

# Arreglos

- ▶ Supongamos que la variable *a* tiene tipo de dato arreglo de int.
- ▶ Podemos exemplificar 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

# 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.

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

Variables				
Nombre	Tipo	Tamaño	Valor	Referencia
a	Lista de int		[5,6,7,8]	B1

# Arreglos Python

- ▶ Se requiere un importar un módulo *array* que permiten utilizar arreglos (otra posibilidad es NumPy).
- ▶ Los arreglos en Python están diseñados para trabajar con datos numéricos (enteros, flotantes, etc.)
- ▶ Al crear el arreglo, se indica su tipo y opcionalmente su contenido inicial.
- ▶ Solo pueden contener elementos de un mismo tipo.

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

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

# Arreglos Python

## ► Tipos de datos del módulo array

Code Type	Python Type	Full Form	Tamaño (en Bytes)
u	unicode character	Python Unicode	2
b	int	Signed Char	1
B	int	Unsigned Char	1
h	int	Signed Short	2
I	int	Signed Long	4
L	int	Unsigned Long	4
q	int	Signed Long Long	8
Q	int	Unsigned Long Long	8
H	int	Unsigned Short	2
f	float	Float	4
d	float	Double	8
i	int	Signed Int	2
l	int	Unsigned Int	2

# Arreglos Python

## Operaciones básicas sobre arrays

Sea  $a$  un array:

$a[i]$  → obtiene el valor del elemento  $i$  de  $a$

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

$a.append(x)$  → añade  $x$  como nuevo elemento de  $a$

$a.remove(x)$  → elimina el primer elemento en  $a$  que coincide con  $x$

$a.index(x)$  → obtiene la posición donde aparece por primera vez el elemento  $x$

$a.count(x)$  → devuelve la cantidad de apariciones del elemento  $x$

$a.insert(p,x)$  → inserta el elemento  $x$  delante de la posición  $p$

# Listas en Python

- En Python, las listas pueden tener elementos de diferentes tipos de datos.
- Son parte de las estructuras de datos integradas de Python, no necesitan importar ningún módulo adicional.
- Al igual que los arrays en Python, tienen tamaño dinámico.
- Son un poco menos eficientes que los arreglos en términos de memoria y rendimiento cuando se trabaja con grandes volúmenes de datos numéricos.

## ¿Cómo se declaran?

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

# Listas en Python

## Operaciones básicas sobre listas

Las listas y los arrays comparten muchas operaciones.

Sea  $a$  una lista:

$a[i]$  → obtiene el valor del elemento  $i$  de  $a$

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

$a.append(x)$  → añade  $x$  como nuevo elemento de  $a$

$a.remove(x)$  → elimina el primer elemento en  $a$  que coincide con  $x$

$a.index(x)$  → obtiene la posición donde aparece por primera vez el elemento  $x$

$a.count(x)$  → devuelve la cantidad de apariciones del elemento  $x$

$a.insert(p, x)$  → inserta el elemento  $x$  delante de la posición  $p$

# Listas en Python

## Operaciones básicas sobre listas

Un ejemplo de una operación que tiene el array y no la lista:

Buffer Info devuelve una tupla con dirección de memoria actual de los arrays y número de elementos (Útil sólo para operaciones de bajo nivel).

```
(operationes_arrays_listas.py > ...
1  # Importar módulo array
2  import array
3
4  # Declarar lista con valores numéricos
5  lista1 = [1, 0, 1, 0, 1, 1, 0, 0]
6
7  # Declarar 'array1' de tipo 'char sin signo' con datos de 'lista1'
8  array1 = array.array('B', lista1)
9
10 print("Buffer info: " + str(array1.buffer_info()))
11 print("Buffer info: " + str(lista1.buffer_info()))
12
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Buffer info: (2071285481920, 8)
Traceback (most recent call last):
  File "c:\@Martin\Facultad\intro-programacion\teoricas\t12 - listas\ejemplos\operaciones_arrays_listas.py", line 11, in <module>
    print("Buffer info: " + str(lista1.buffer_info()))
                                         ^
AttributeError: 'list' object has no attribute 'buffer_info'
```

# Listas en Python

## Como recorrer una lista

Podemos utilizar las distintas estructuras de control de ciclo para recorrer los elementos de una lista utilizando índices:

```
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
```

# Listas en Python

## Como recorrer una lista

También podremos utilizar el for para recorrer directamente sus elementos:

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

# Recorriendo los elementos
for edad in edades:
    print(edad)
```

# Listas en Python

## Matrices

Podemos pensar una matriz como una lista de listas. Podemos recorrerlas a través de sus índices:

```
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 indices ++")  
# 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
```

# Listas en Python

## Iterables

En Python, aquellas variables cuyo tipo de dato sea *iterable* pueden ser recorridas con un for.

NOTA (para quienes saben Python): los conceptos de iterable e iterators están fuera del temario de la materia.

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

# Recorriendo los elementos
for edad in edades:
    print(edad)
```

# Tipos Abstractos de Datos

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

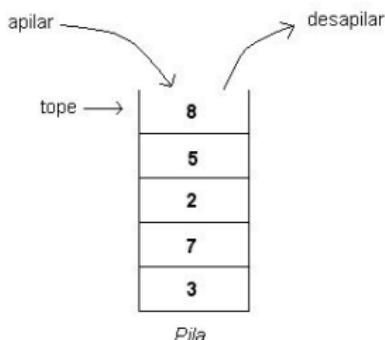
El tipo lista que estuvimos viendo es un TAD:

- ▶ 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

Una pila es una lista de elementos de la cual se puede extraer el último elemento insertado.

- ▶ También se conocen como listas LIFO (Last In - First Out / el último que entra es el primero que sale)
- ▶ Operaciones básicas
  - ▶ apilar: ingresa un elemento a la pila
  - ▶ desapilar: saca el último elemento insertado
  - ▶ tope: devuelve (sin sacar) el ultimo elemento insertado
  - ▶ vacia: retorna verdadero si está vacía



# Pila

Ejemplos de problemas que naturalmente se modelarían con una pila

- ▶ Una pila de platos acumulados en la bacha esperando a ser lavados (no se puede sacar los de abajo, sin antes lavar los ultimos apoyados).
- ▶ Una pila de libros o meter y sacar libros de una caja.
- ▶ En las góndolas del supermercado, el fondo del estando es el fondo de la pila, y el tope son los artículos que se pueden tomar facilmente.
- ▶ Ponerse muchas remeras, una arriba de la otra.

# Pila

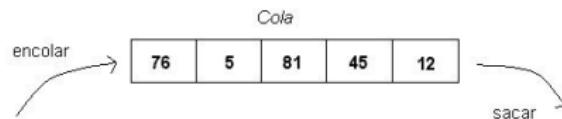
En Python, el tipo lista provee los métodos necesarios para poder usar una lista como una pila

- ▶ 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 lista de elementos en donde siempre se insertan nuevos elementos al final de la lista y se extraen elementos desde el inicio de la lista.

- ▶ También se conocen como listas FIFO (First In - First Out / el primero que entra es el primero que sale)
- ▶ Operaciones básicas
  - ▶ encolar: ingresa un elemento a la cola
  - ▶ sacar: saca el primer elemento insertado
  - ▶ vacia: retorna verdadero si está vacía



# Cola

Ejemplos de problemas que naturalmente se modelarían con una cola

- ▶ La fila en la parada de colectivos
- ▶ La fila de la caja de un supermercado
- ▶ La fila en la cabina de peaje

# Cola

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`