



**Universidad  
Interamericana de Panamá**  
LAUREATE INTERNATIONAL UNIVERSITIES\*

**Carrera:**

Ing. De Sistemas Computacionales

**Alumno:**

Urbano Igualá, 9-744-1120

**Profesor:**

Prof. Leonardo Esqueda

**Materia:**

Estructura de Datos II

**Tema:**

Tarea Inicial

**Fecha de entrega**

25 de septiembre del 2022

**Matriz:** Una matriz es un vector que contiene elementos homogéneos, es decir, que pertenecen al mismo tipo de datos. Los elementos se asignan con ubicaciones de memoria contiguas que permiten una fácil modificación, es decir, agregar, eliminar, acceder a los elementos. En Python, tenemos que usar el array módulo para declarar matrices. Si los elementos de una matriz pertenecen a diferentes tipos de datos, se lanza una excepción "Tipos de datos incompatibles".

```
import array

sample_array = array.array('i', [1, 2, 3])

for i in sample_array:
    print(i)
```

**Salida:**

```
1
2
3
```

#### Listas(Arreglos)

Puede constar de elementos pertenecientes a diferentes tipos de datos

No es necesario importar explícitamente un módulo para la declaración

No puede manejar operaciones aritméticas directamente

Se puede anidar para contener diferentes tipos de elementos.

Preferido para una secuencia más corta de elementos de datos

Una mayor flexibilidad permite una fácil modificación(adición, eliminación) de datos

La lista completa se puede imprimir sin ningún bucle explícito

Consume más memoria para agregar elementos fácilmente

#### Matriz

Solo consta de elementos pertenecientes al mismo tipo de datos

Necesidad de importar explícitamente un módulo para la declaración

Puede manejar directamente operaciones aritméticas

Debe contener todos los elementos anidados del mismo tamaño

Preferido para una secuencia más larga de elementos de datos

Menos flexibilidad desde la adición, la eliminación debe realizarse en función de los elementos

Se debe formar un bucle para imprimir o acceder a los componentes de la matriz

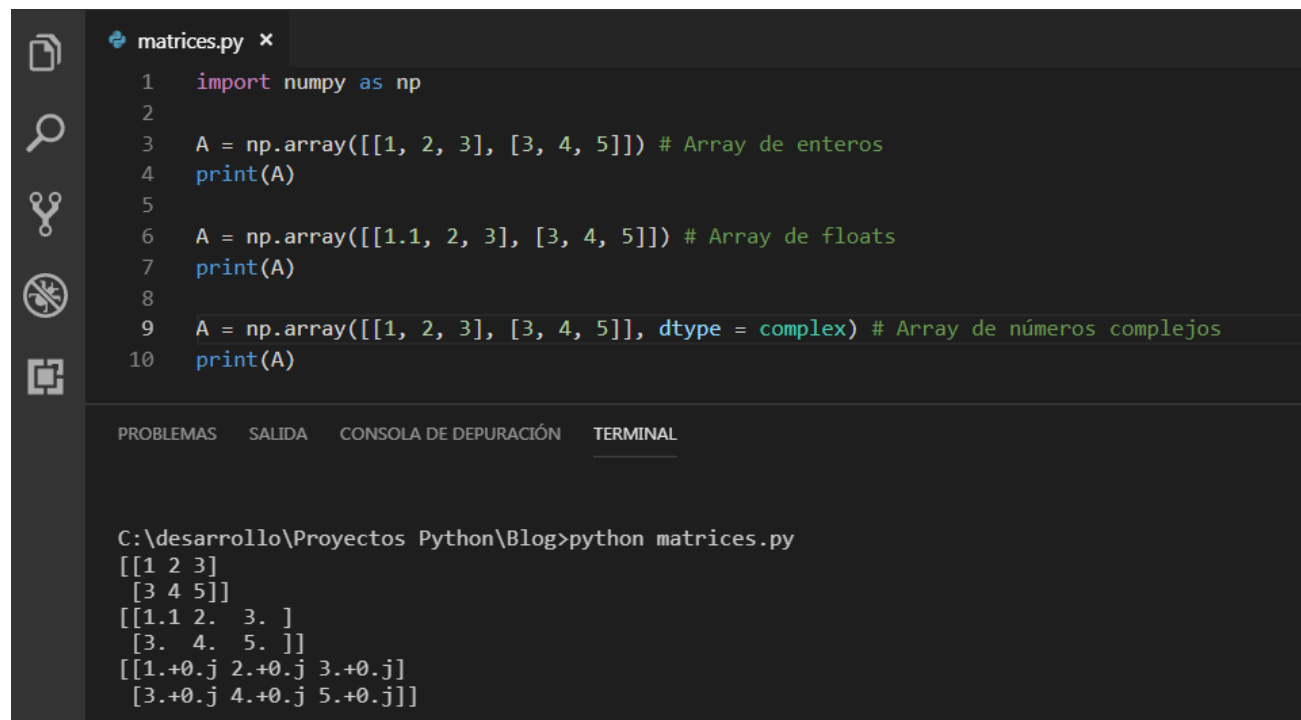
Comparativamente más compacto en tamaño de memoria

Arreglos Ejemplo:

```
>>> from numpy.random import random

>>> random(3)
array([ 0.53077263,  0.22039319,  0.81268786])
>>> random(3)
array([ 0.07405763,  0.04083838,  0.72962968])
>>> random(3)
array([ 0.51886706,  0.46220545,  0.95818726])
```

Matriz Ejemplo:

The image shows a code editor window with a file named 'matrices.py'. The code defines three NumPy arrays: a 2x3 integer array, a 2x3 float array, and a 2x3 complex array. The terminal output shows the execution of the script, displaying the arrays as they are printed. The code is as follows:

```
1 import numpy as np
2
3 A = np.array([[1, 2, 3], [3, 4, 5]]) # Array de enteros
4 print(A)
5
6 A = np.array([[1.1, 2, 3], [3, 4, 5]]) # Array de floats
7 print(A)
8
9 A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex) # Array de números complejos
10 print(A)
```

The terminal output is:

```
C:\desarrollo\Proyectos Python\Blog>python matrices.py
[[1 2 3]
 [3 4 5]]
[[1.1 2.  3. ]
 [3.  4.  5. ]]
[[1.+0.j 2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j 5.+0.j]]
```

## Diferencias entre Pilas y Colas

Stack y Queue son ambas estructuras de datos no primitivas. Las principales diferencias entre la pila y la cola son que la pila utiliza el método LIFO (último en entrar, primero en salir) para acceder y agregar elementos de datos, mientras que la cola usa el método FIFO (primero en entrar, primero en salir) para acceder y agregar elementos de datos.

La pila solo tiene un extremo abierto para empujar y hacer estallar los elementos de datos en la otra parte. La cola tiene ambos extremos abiertos para encolar y sacar en cola los elementos de datos.

La pila y la cola son las estructuras de datos utilizadas para almacenar elementos de datos y en realidad se basan en algún equivalente del mundo real. Por ejemplo, la pila es una pila de CD donde puede sacar y poner un CD en la parte superior de la pila de CD. De manera similar, la cola es una cola para los boletos de teatro donde la persona que se encuentra en primer lugar, es decir, se colocará primero en la parte delantera de la cola y la nueva persona que llegue aparecerá en la parte posterior de la cola (parte posterior de la cola).

### Gráfica comparativa

Bases para la comparación	Apilar	Cola
Principio de funcionamiento	LIFO (Último en Primero en salir)	FIFO (Primero en Primero en salir)
Estructura	El mismo extremo se utiliza para insertar y eliminar elementos.	Un extremo se utiliza para la inserción, es decir, el extremo trasero y el otro extremo se utiliza para la eliminación de elementos, es decir, el extremo delantero.
Número de punteros utilizados	Uno	Dos (en caso de cola simple)
Operaciones realizadas	Push and Pop	Encolado y encolado
Examen de la condición de vacío	Arriba == -1	Frente == -1    Delantero == Trasero + 1
Examen de estado completo	Top == Max - 1	Trasero == Max - 1
Variantes	No tiene variantes.	Tiene variantes como cola circular, cola de prioridad, cola de doble finalización.

Implementación	Más simple	Comparativamente complejo
----------------	------------	---------------------------

### Diferencias clave entre la pila y la cola

1. Por otro lado, la pila sigue el mecanismo LIFO. La cola sigue el mecanismo FIFO para agregar y eliminar elementos.
2. En una pila, el mismo extremo se utiliza para insertar y eliminar los elementos. Por el contrario, se utilizan dos extremos diferentes en la cola para insertar y eliminar los elementos.
3. Como la pila solo tiene un extremo abierto, es la razón por la que se usa un solo puntero para referirse a la parte superior de la pila. Pero la cola utiliza dos punteros para referirse al frente y al final de la cola.
4. Stack realiza dos operaciones conocidas como push y pop, mientras que en la cola se conoce como enqueue y dequeue.
5. La implementación de la pila es más fácil, mientras que la implementación de la cola es difícil.
6. La cola tiene variantes como la cola circular, la cola de prioridad, la cola de doble finalización, etc. Por el contrario, la pila no tiene variantes.

### Ejemplo Pila:

```
int stack [5], top = -1;
void push()
{
int item;
if (top < 4)
{
printf ("Enter the number") ;
scan ("%d", & item) ;
top = top + 1;
stack [top] = item;
}
else
{
printf (" Stack is full");
}
}
```

```

int stack [5], top = -1;
void pop()
{
    int item;
    if (top >= 4)
    {
        item = stack [top];
        top = top - 1;
        printf ("Number deleted is = %d", item) ;
    }
    else
    {
        printf (" Stack is empty");
    }
}

```

#### Ejemplo Cola:

```

int queue [5], Front = -1 and rear = -1;
void add ()
{
    int item;
    if ( rear < 4)
    {
        printf ("Enter the number") ;
        scan ("%d", & item) ;
        if (front == -1)
        {
            front =0 ;
            rear =0 ;
        }
        else
        {
            rear = rear + 1;
        }
        queue [rear] = item ;
    }
    else
    {
        printf ("Queue is full") ;
    }
}

```

## Como Crear una Estructura de Lista

Para crear una lista en Python, debemos **colocar todos los elementos** que queramos añadir a esa lista entre corchetes [ ] y separados por comas.

```
myList = ["pan", "leche", "azúcar"]
```

Las listas pueden contener cuantos **elementos** queramos y como ya decíamos, pueden ser de diferentes tipos (enteros, cadenas, booleanos, etc.)

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True]
```

Las listas también pueden **contener otras listas**, las cuales contarán como un único valor dentro de la lista.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
```

Para poder ver la lista creada o un elemento en concreto de la lista, podemos usar el **índice**, siempre conociendo que el primer elemento de la lista ocupa la posición 0.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print("Lista completa ")
print(myList)
print("Solo el valor que ocupa la posición 2 ")
print(myList[2])
```

```
Lista completa
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
Solo el valor que ocupa la posición 2
azúcar
```

## Métodos de Lista

### Len ()

Este método nos devuelve la **longitud de un objeto**, en este caso al usarlo sobre la lista, nos devolverá el número de datos que la componen.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print(len(myList)) #Devuelve el número de elementos de la lista, 8
```

## Index()

Devuelve la **posición del elemento** que hayamos introducido para su búsqueda, si hubiera dos elementos iguales, solo devolvería la posición del primero.

```
print(myList.index("azúcar")) #Devuelve la posición de la primera aparición de azúcar en la lista
```

## Append()

Este método añade un **elemento al final de la lista**. En el ejemplo vemos la lista antes y después de añadir el nuevo dato, en nuestro caso hemos añadido de nuevo el dato “leche” y una lista.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print(myList)
myList.append("leche")
print(myList)
myList.append([4, 5])
print(myList)
```

```
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99, 'leche']
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99, 'leche', [4, 5]]
```

## Count()

Con este método podemos saber el **número de veces que se repite un elemento en nuestra lista**.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99, "leche"]
print(myList.count("leche"))
#Nos devuelve 2, que es el número de veces que se repite leche en la lista
```

## Extend()

Con este método podemos **añadir elementos de una lista a otra** pero a diferencia de append(), como nuevos elementos dentro de la nueva lista.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print(myList)
myList.extend([4, 5])
print(myList)
```

```
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99, 4, 5]
```



## Remove ()

Con este método podemos **borrar un elemento de la lista** indicándolo como argumento. Remove() solo borrará la primera aparición del elemento.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print(myList)
myList.remove(2)
print(myList)
```

```
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
['pan', 'leche', 'azúcar', 4.6, True, ['Javi', 'María'], 99]
```

## Insert()

Con este método podemos añadir un elemento a nuestra lista pero a diferencia de append(), podemos **insertarlo en la posición que queramos dentro del índice**.

```
myList = ["pan", "leche", "azúcar", 2, 4.6, True, ["Javi", "María"], 99]
print(myList)
myList.insert(2, "Burro") #Primero se indica la posición y luego el dato
print(myList)
```

```
['pan', 'leche', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
['pan', 'leche', 'Burro', 'azúcar', 2, 4.6, True, ['Javi', 'María'], 99]
```

### Fuentes:

<https://es.gadget-info.com/difference-between-stack>

<https://www.programarya.com/Cursos/Python/estructuras-de-datos/tablas>

<https://pythondiario.com/2019/01/matrices-en-python-y-numpy.html>

[https://eiposgrados.com/blog-python/crear-una-lista-en-python/#:~:text=Para%20crear%20una%20lista%20en,cadenas%2C%20booleanos%2C%20etc.\)](https://eiposgrados.com/blog-python/crear-una-lista-en-python/#:~:text=Para%20crear%20una%20lista%20en,cadenas%2C%20booleanos%2C%20etc.))