Lógica de programación I

Juan Pablo Restrepo Uribe

Ing. Biomedico

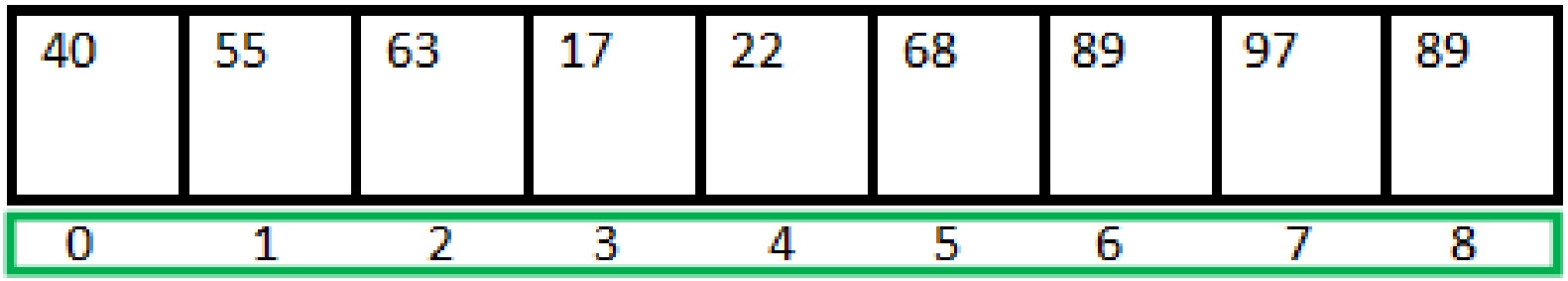
MSc. Automatización y Control Industrial

jprestrepo@correo.iue.edu.co

Institución Universitaria de Envigado

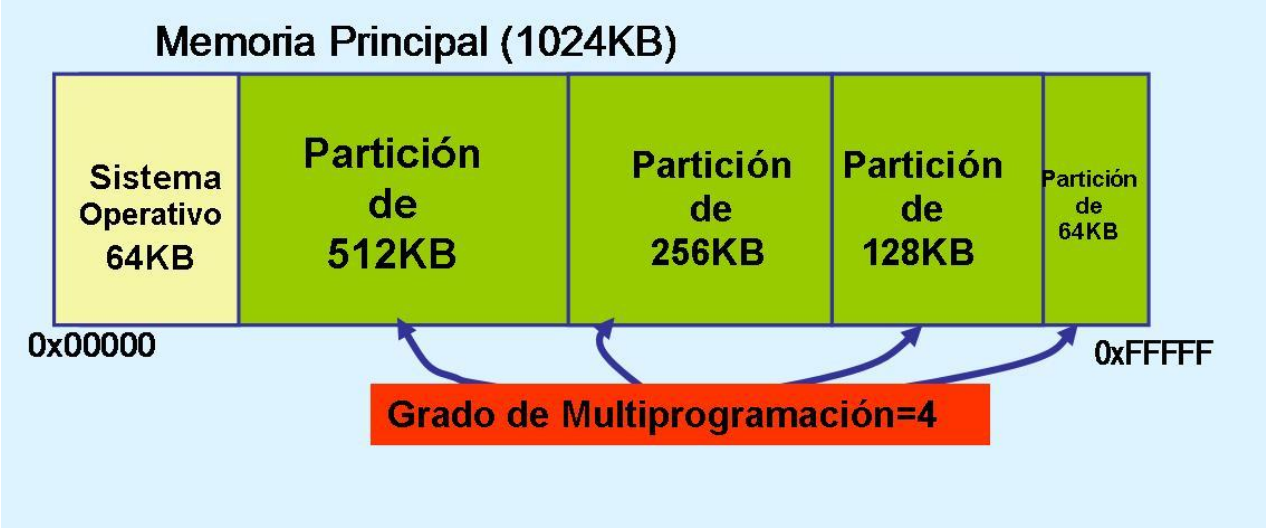
Array

Es una colección de elementos del **mismo tipo** que se almacenan en una secuencia contigua de memoria. Se puede acceder a cada elemento del array mediante un índice que indica su posición dentro del array.



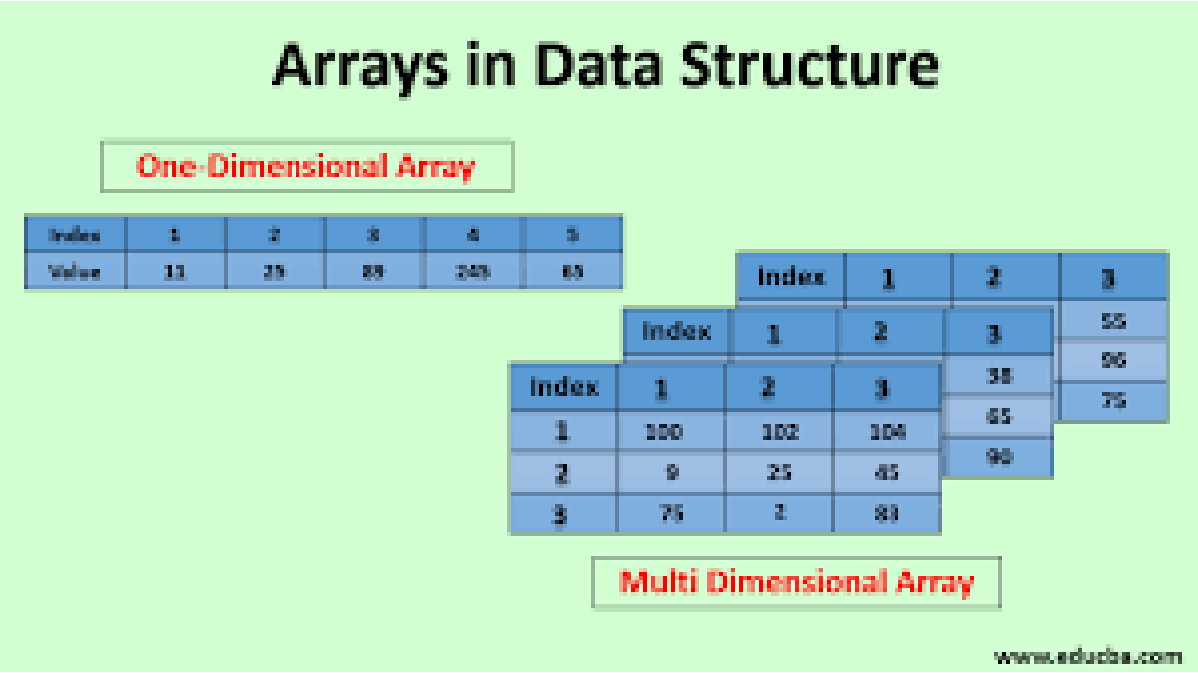
Array

Son estructuras de datos fundamentales y se utilizan ampliamente en programación debido a su eficiencia en el acceso a los elementos y su implementación. Sin embargo, los arrays tienen limitaciones en cuanto a su tamaño fijo



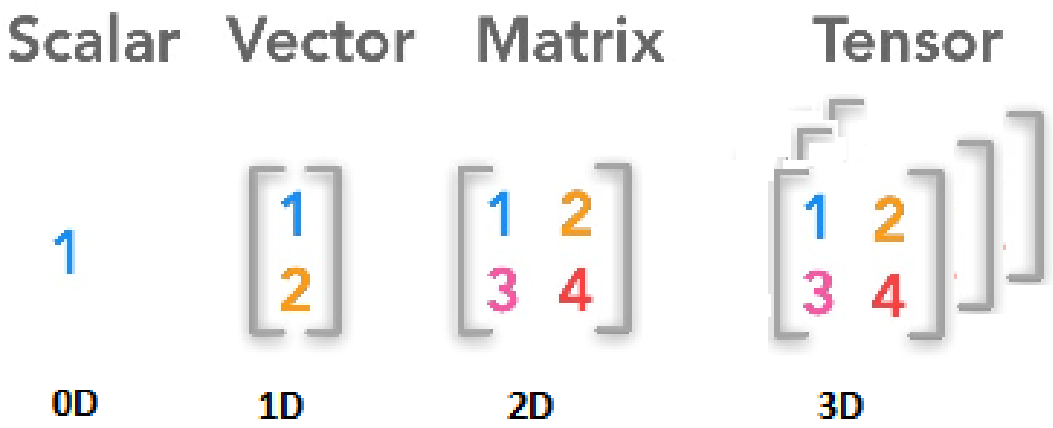
Array

Los arrays pueden ser unidimensionales, bidimensionales o multidimensionales, dependiendo de la cantidad de índices necesarios para acceder a un elemento específico.



Array

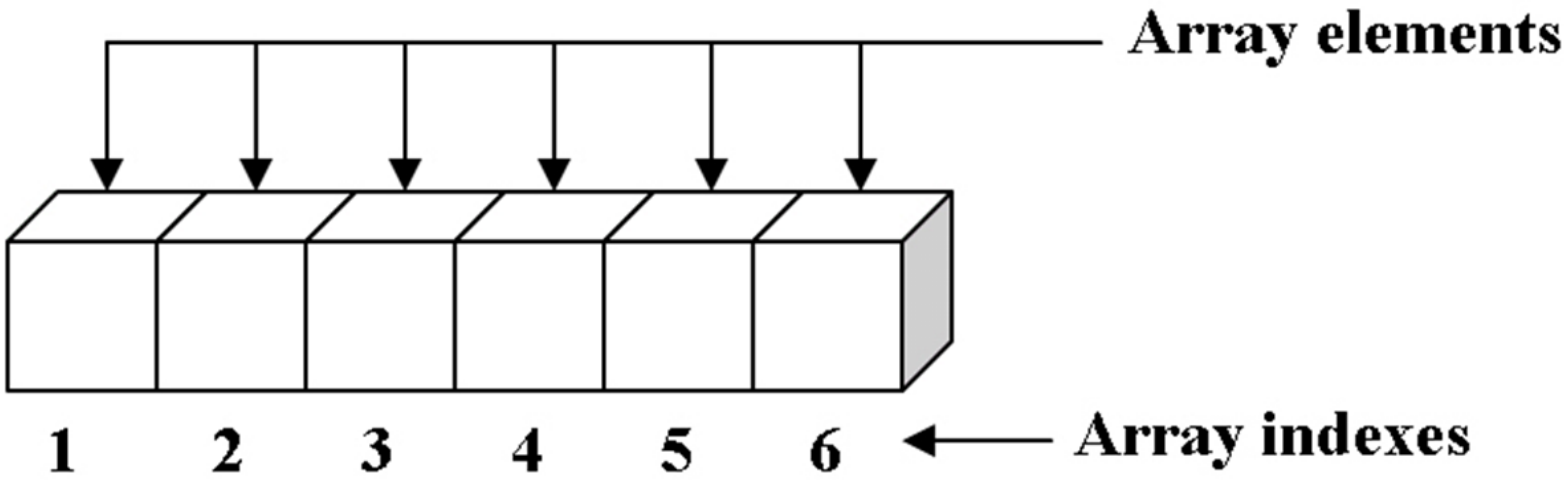
- Un array unidimensional podría ser una lista de números enteros como [1, 2, 3, 4, 5], donde cada elemento se puede acceder mediante un solo índice
- Un array bidimensional podría ser una tabla de números como [[1, 2], [3, 4], [5, 6]], donde se necesitan dos índices para acceder a un elemento específico (por ejemplo, array[1][0] para acceder al elemento en la segunda fila y primera columna).



Array (Características)

- **Almacenamiento contiguo:** Los elementos se almacenan en posiciones de memoria contiguas.
- **Tamaño fijo:** El tamaño se determina en el momento de su creación y no puede cambiar dinámicamente.
- **Acceso aleatorio:** Los elementos se pueden acceder directamente utilizando un índice entero que indica su posición en el array.
- **Eficiencia en el acceso:** El acceso a los elementos de un array es generalmente muy eficiente y tiene un tiempo de acceso constante O(1).
- **Homogeneidad:** Los elementos deben ser del mismo tipo de datos.
- **Eficiencia en el uso de la memoria:** Ocupan un espacio de memoria contiguo y el tamaño es fijo.
- **Facilidad de iteración:** Permiten una fácil iteración a través de sus elementos utilizando bucles.

Array (Características)



One-dimensional array with six elements

Array – Ventajas

- **Acceso Aleatorio Eficiente:** El acceso a elementos individuales es muy eficiente, ya que se puede acceder directamente a cualquier elemento mediante su índice.
- **Uso de Memoria Más Eficiente:** Son más eficientes en términos de uso de memoria en comparación con algunas implementaciones de listas, especialmente las listas enlazadas que requieren nodos adicionales para almacenar referencias.
- **Operaciones de Búsqueda y Ordenación más Rápidas:** Son más adecuados para operaciones de búsqueda y ordenación, especialmente cuando se tiene una gran cantidad de datos.
- **Iteración Directa:** Es más simple y eficiente que iterar sobre una lista enlazada, ya que no es necesario seguir punteros para acceder a los elementos siguientes.

Array (Desventajas)

No podemos ajustar su tamaño a la mitad de la ejecución de un programa

¿creamos uno nuevo con la nueva dimensión deseada?

2 opciones

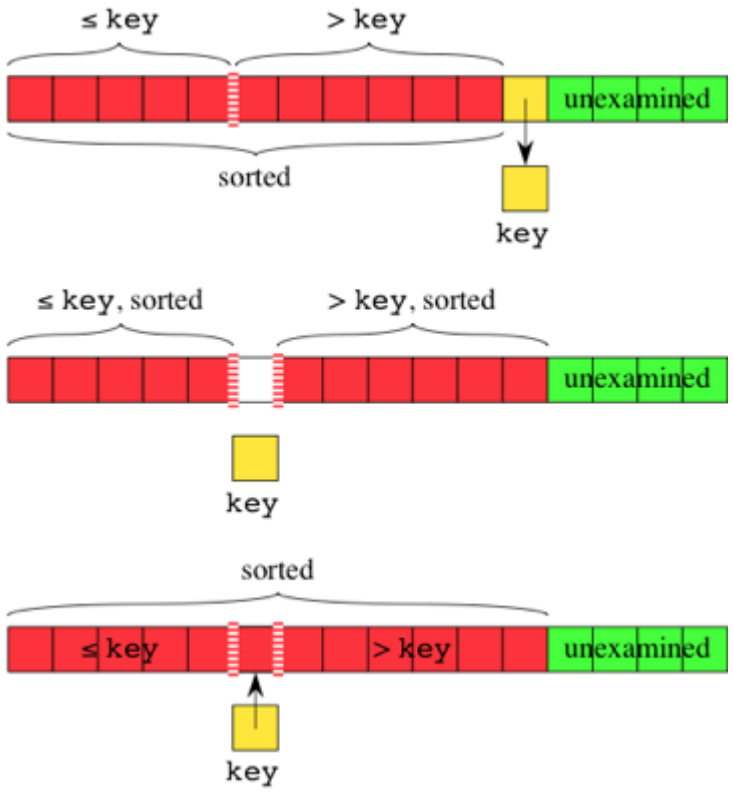
- Crear un array lo suficientemente grande para almacenar nuestros datos, pero esto deriva en un **desperdicio de memoria totalmente innecesario**
- Crear un nuevo array, doblar el tamaño de éste cada vez que se necesite crecer y copiar los datos del array anterior al nuevo array. Esto tiene el mismo nivel de complejidad que si tuviéramos un array único suficientemente grande, pero con la ventaja de que sólo va a crecer cuándo sea necesario, evitando así desperdiciar memoria.

Array - Ventajas – Sobrecarga de memoria

Es un problema que puede surgir cuando se asigna más memoria de la necesaria para almacenar los datos. Esto puede ocurrir debido a la manera en que se diseñan y gestionan las estructuras de datos, así como a errores en el código que las manipula. La sobrecarga de almacenamiento puede provocar un uso ineficiente de la memoria del sistema y afectar negativamente al rendimiento del programa.

Array - Desventajas

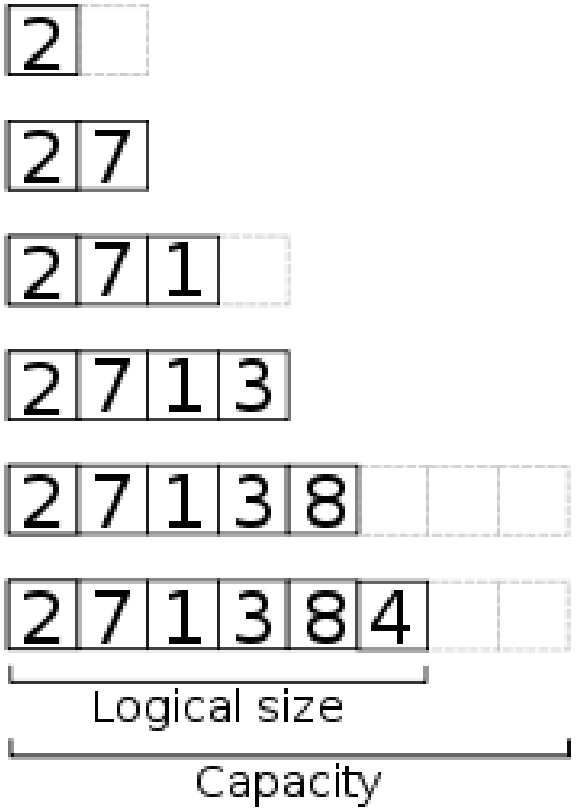
- **Tamaño fijo:** Su tamaño generalmente es fijo y debe especificarse al crear el array. Esto puede ser problemático si no se conoce de antemano la cantidad exacta de elementos que se necesitarán
- **Inserción y eliminación ineficientes:** Insertar o eliminar elementos en el medio de un array puede ser ineficiente, especialmente si se necesitan mantener los elementos en orden. Esto puede requerir desplazar todos los elementos siguientes para hacer espacio para el nuevo elemento o para cerrar el espacio dejado por el elemento eliminado.



Array - Desventajas

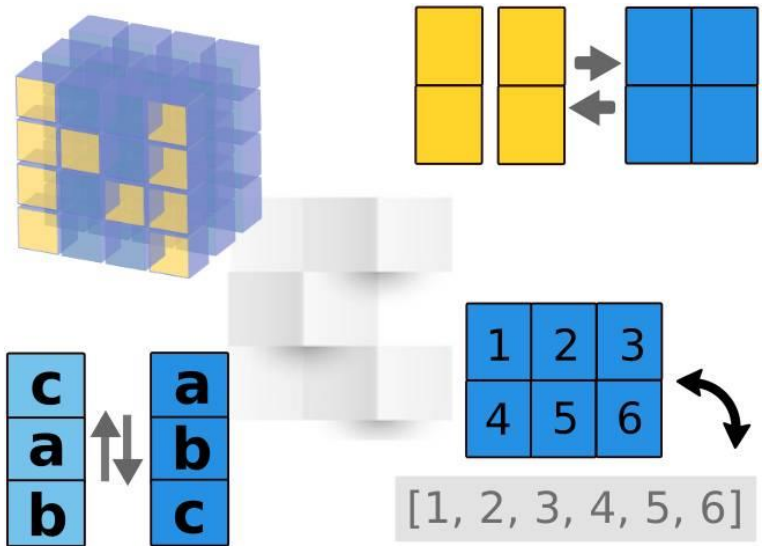
- **Uso ineficiente de la memoria con tamaño no utilizado:** Si se reserva un array con un tamaño mayor al necesario, puede haber una porción de memoria reservada, pero sin utilizar
- **homogeneidad de tipo:** En muchos lenguajes de programación, los arrays requieren que todos sus elementos sean del mismo tipo.
- **Redimensionamiento costoso:** Cambiar el tamaño de un array puede ser una operación costosa en términos de tiempo de ejecución, ya que puede requerir la creación de un nuevo array con el tamaño deseado y la copia de todos los elementos del array original al nuevo array.

Array



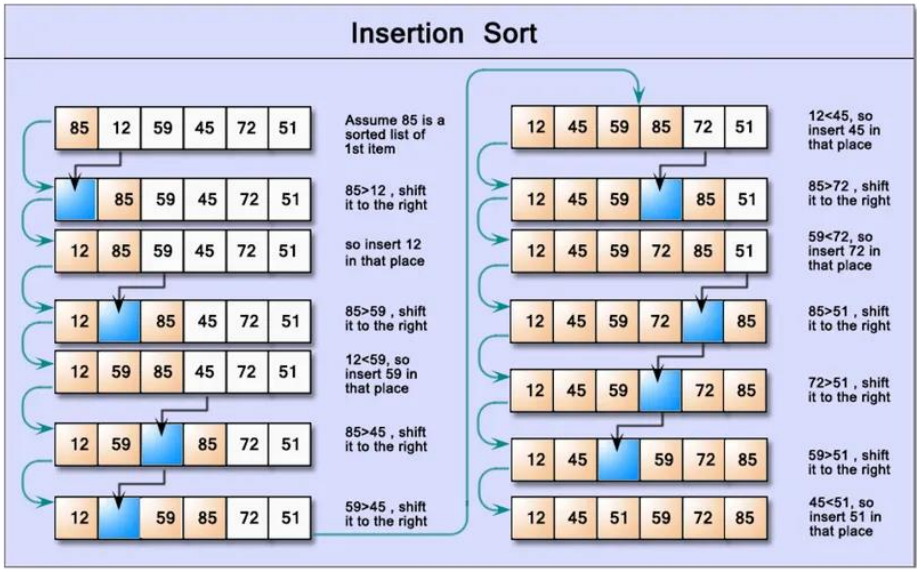
Array

Cada elemento en un array tiene una posición única dentro de la estructura, que se identifica mediante un índice. Este índice representa la ubicación del elemento en el array y suele ser un número entero no negativo. Con el índice, puedes acceder, modificar o eliminar un elemento específico del array.



Array

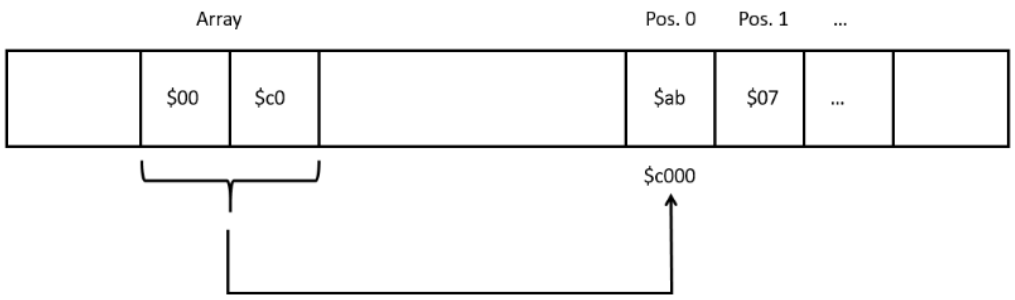
En la mayoría de los lenguajes de programación, los índices de los arrays no ocupan espacio de memoria adicional cuando el array se crea. Los índices simplemente son valores enteros que se utilizan para acceder a los elementos del array, y estos valores se almacenan en registros o en la pila de la memoria del programa durante la ejecución del programa.



Array

En Python, los índices de los arrays son simplemente valores enteros que se utilizan para acceder a los elementos del array. Estos valores enteros están implícitos en el lenguaje y no necesitan ser almacenados explícitamente.

Es importante tener en cuenta que algunos lenguajes de programación, como C o C++, permiten acceder a los elementos de los arrays mediante punteros aritméticos, lo que significa que los índices se pueden representar como desplazamientos de memoria desde la dirección base del array.



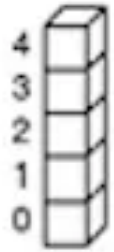
Array

Otra estructura muy común que nos ayuda a representar datos reales son las llamadas matrices o tablas que no son más que arrays de múltiples dimensiones.

Si un arreglo es una colección consecutiva de valores, se puede decir que son de tipo lineal en una dimensión plana ó uni-dimensionales, si quisiéramos formar una matriz o tabla (matrix o table) podríamos crear un array bi-dimensional, lo que quiere decir que tendremos que localizar a un elemento por su index [i , j], o bien su ubicación lineal a la derecha y hacia abajo como en un plano cartesiano con dimensiones xy asemejando a una tabla, o podríamos crear un array tri-dimensional, agregando una dimensión más de “fondo”, que haría que localizáramos a un elemento en su index [i, j, k], en plano cartesiano como xyz.

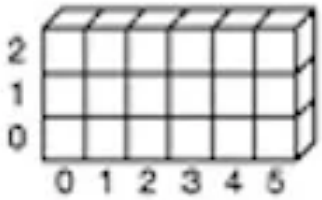
Array

One-Dimensional Arrays

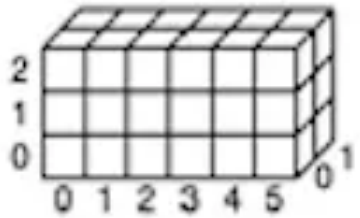


One-Dimensional
int[5]

Rectangular Arrays

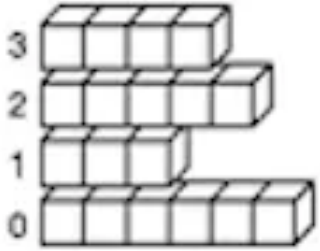


Two-Dimensional
int[3,6]



Three-Dimensional
int[3,6,2]

Jagged Arrays



Jagged Array
int[4][]

Array

```python
class MiArray:
    def __init__(self, capacidad):
        self.capacidad = capacidad
        self.size = 0
        self.data = [None] * capacidad

    def __getitem__(self, index):
        return self.data[index]

    def __setitem__(self, index, value):
        self.data[index] = value

    def append(self, value):
        if self.size < self.capacidad:
            self.data[self.size] = value
            self.size += 1
        else:
            print("El array está lleno, no se puede agregar más elementos.")

    def __len__(self):
        return self.size

    def __str__(self):
        return str(self.data[:self.size])
```

Array (Métodos magicos)

- **__getitem__(self, index):** Este método se llama cuando accedes a un elemento del objeto utilizando la sintaxis de corchetes [ ], por ejemplo, objeto[indice]. El parámetro index representa el índice del elemento que se está accediendo.
- **__setitem__(self, index, value):** Este método se llama cuando asignas un valor a un elemento del objeto utilizando la sintaxis de corchetes [ ], por ejemplo, objeto[indice] = valor. Los parámetros index y value representan el índice del elemento que se está asignando y el valor que se está asignando, respectivamente.

Array

NumPy es una biblioteca fundamental en Python utilizada principalmente para el cálculo numérico y la manipulación de matrices multidimensionales o Arrays. Proporciona un conjunto de funciones y herramientas para realizar operaciones matemáticas complejas de manera eficiente y fácil en estos Arrays

Array

1. **Arrays NumPy**: Son estructuras de datos fundamentales en NumPy. Son similares a las listas de Python, pero con la capacidad de manejar operaciones matemáticas eficientes en grandes conjuntos de datos.
2. **Velocidad**: NumPy está escrito en C y Fortran, lo que le proporciona una velocidad significativa sobre las operaciones numéricas en Python puro. Esto es crucial para el procesamiento eficiente de grandes conjuntos de datos.
3. **Funciones Matemáticas**: NumPy proporciona una amplia gama de funciones matemáticas que pueden ser aplicadas a los arrays NumPy, como trigonometría, álgebra lineal, estadísticas, entre otros.
4. **Indexación y Slicing Avanzados**: NumPy permite indexar y rebanar (slice) arrays de manera muy poderosa, lo que facilita la manipulación y análisis de datos.
5. **Integración con otras bibliotecas**: NumPy se utiliza ampliamente en conjunto con otras bibliotecas de Python para ciencia de datos y computación científica, como pandas, SciPy, Matplotlib, entre otras.