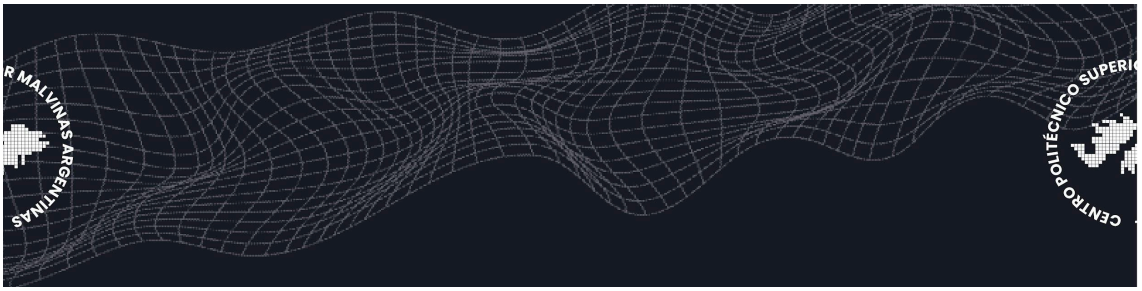


Nombre del bloque PROGRAMACIÓN 1

Año de cursada 1º AÑO

Clase Nº 7: Estructura de datos.

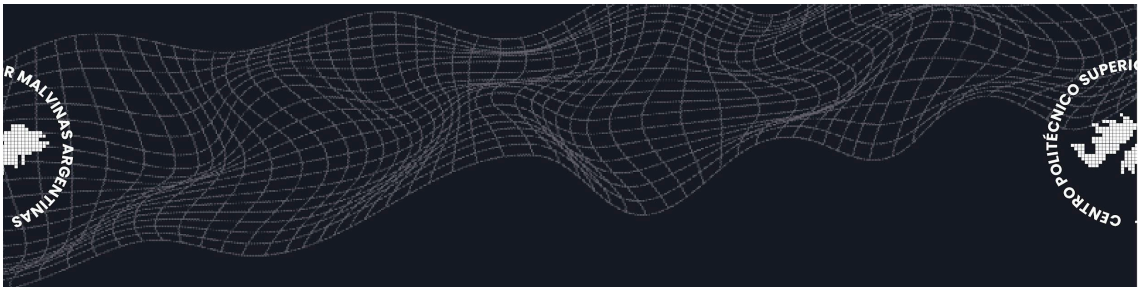
Contenido

1. Contenido

- Introducción a las estructuras de datos
- Arrays unidimensionales (vectores)
- Operaciones con vectores: Asignación, Lectura/Escritura de datos, Acceso secuencial, Actualización
- Arrays de varias dimensiones: bidimensionales (tablas/matrices) y multidimensionales
- Almacenamiento de arrays en memoria
- Estructuras de datos dinámicas en Python: Listas (list), Tuplas (tuple), Conjuntos (set), Diccionarios (dict)

2. Presentación

¡Bienvenidos a la séptima clase de Programación 1! Hoy exploramos las estructuras de datos, fundamentales para organizar y almacenar información eficientemente en



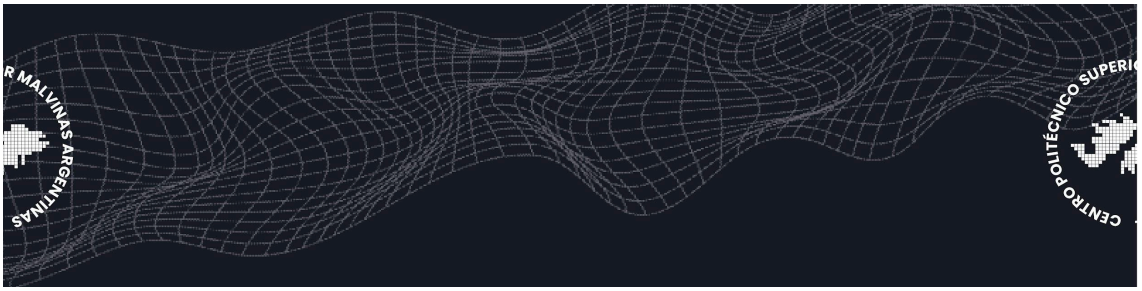
programación. Comenzaremos con los arrays unidimensionales, conocidos como vectores, y sus operaciones básicas. Luego, avanzaremos a estructuras más complejas como arrays bidimensionales y multidimensionales. Finalmente, exploraremos las estructuras de datos dinámicas en Python: listas, tuplas, conjuntos y diccionarios.

3. Desarrollo

Arrays (arreglos) unidimensionales: *los vectores.*

Los arrays unidimensionales, comúnmente conocidos como vectores, son una estructura de datos fundamental en programación que se utiliza para almacenar una colección ordenada de elementos del mismo tipo. Estos elementos se almacenan de manera consecutiva en la memoria, lo que permite un acceso eficiente a través de índices numéricos.

Ejemplo de un vector en Python:



mi_vector = [1, 2, 3, 4, 5]

En este ejemplo, mi_vector es un array unidimensional que contiene cinco elementos enteros. Puedes acceder a cada elemento del vector utilizando un índice numérico. Por ejemplo:

En Python:

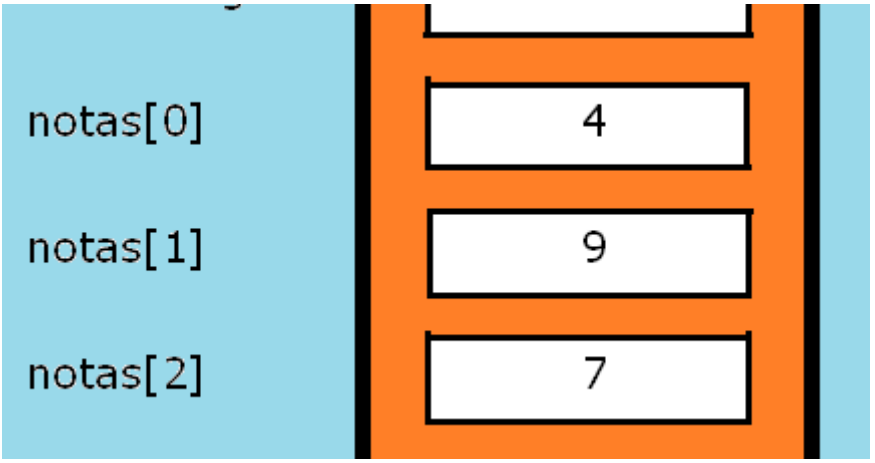
primer_elemento = mi_vector[0] # Acceder al primer elemento (1)

segundo_elemento = mi_vector[1] # Acceder al segundo elemento (2)

Los vectores son útiles para almacenar datos cuando se necesita una estructura de datos simple y rápida para acceder a los elementos. Son especialmente útiles cuando se trabaja con datos homogéneos, es decir, datos del mismo tipo (en este caso, enteros). Además, se pueden realizar una serie de operaciones comunes en vectores, como asignar valores a los elementos, leer y escribir datos, recorrer todo el vector para realizar operaciones en cada elemento, y actualizar los valores existentes.

En resumen, los vectores (o arrays unidimensionales) son una estructura de datos esencial en programación que permite almacenar y acceder eficientemente a una colección ordenada

de elementos del mismo tipo, y son ampliamente utilizados en una variedad de aplicaciones informáticas.



La imagen anterior, ilustra un vector para almacenar notas de alumnos. Imagine tener una variable de tipo primitiva por cada nota.

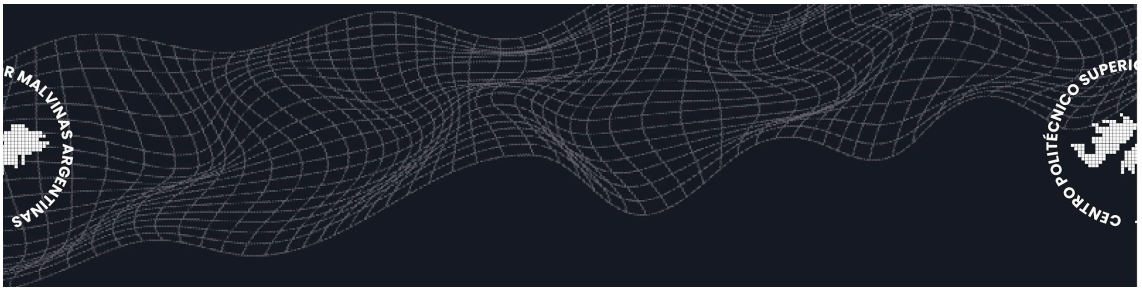
Operaciones con Vectores en Programación

Asignación de Valores a un Vector:

La asignación es la operación básica que permite establecer valores en un vector. Puedes asignar un valor específico a un elemento del vector utilizando su índice.

En Python:

```
mi_vector = [1, 2, 3, 4, 5]
```




```
mi_vector[2] = 7  # Asigna el valor 7 al tercer elemento del vector
```

Lectura/Escritura de Datos

La lectura implica acceder a los valores almacenados en un vector. La escritura implica modificar esos valores. Puedes leer y escribir datos en un vector utilizando índices.

Por ejemplo:

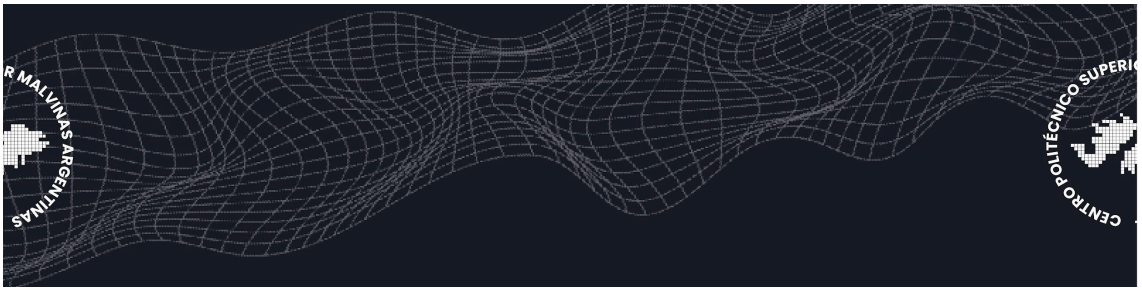
Python:

```
valor = mi_vector[1]  # Lectura del segundo elemento del vector (2)

mi_vector[3] = 10     # Escritura en el cuarto elemento del vector
```

Acceso Secuencial al Vector (Recorrido):

Una operación común es recorrer (o iterar) un vector para procesar todos sus elementos. Puedes utilizar bucles, como un bucle for en Python, para acceder secuencialmente a los elementos de un vector y realizar acciones en cada uno de ellos. Aquí tienes un ejemplo de recorrido de un vector en Python:



for elemento in mi_vector:

Realiza una acción en cada elemento (por ejemplo, imprimirlo)

print(elemento)

Actualización de un Vector

La actualización implica modificar uno o varios elementos del vector. Esto se puede hacer utilizando los índices y asignando nuevos valores a esos elementos. Por ejemplo, si deseas aumentar todos los elementos de un vector en 1:

En Python

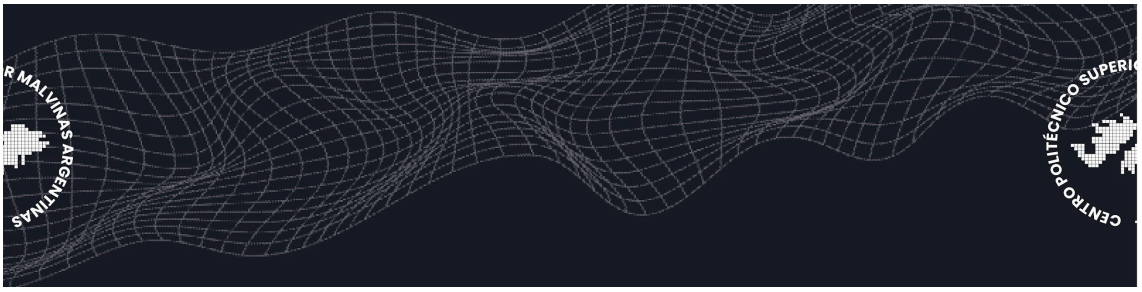
for i in range(len(mi_vector)):

mi_vector[i] += 1 # Aumenta cada elemento en 1

Estas operaciones son esenciales cuando trabajas con vectores (o arrays) en programación. Te permiten manipular y gestionar los datos almacenados en el vector de acuerdo con tus necesidades específicas.

Arrays multidimensional

Los arrays de varias dimensiones son una extensión de los vectores unidimensionales y permiten almacenar datos en una



estructura más compleja con múltiples niveles de profundidad. Entre los tipos de arrays de varias dimensiones, se encuentran los arrays bidimensionales

(también conocidos como tablas o matrices) y los arrays multidimensionales.

Arrays Bidimensionales (Tablas o Matrices)

Los arrays bidimensionales son estructuras de datos en las que los elementos se organizan en filas y columnas, formando una especie de tabla o matriz. Se utilizan comúnmente para representar datos tabulares, como hojas de cálculo o tablas de bases de datos.

Ejemplo de una matriz 2x3 en Python:

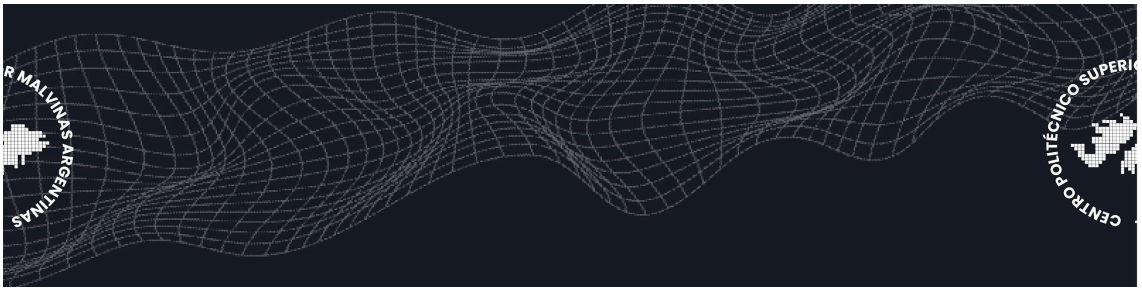
```
Python:

matriz = [

[1, 2, 3],

    [4, 5, 6]

]
```



En este ejemplo, la matriz tiene dos filas y tres columnas. Puedes acceder a elementos individuales especificando el índice de fila y columna, por ejemplo, matriz[1][2] para acceder al valor 6.

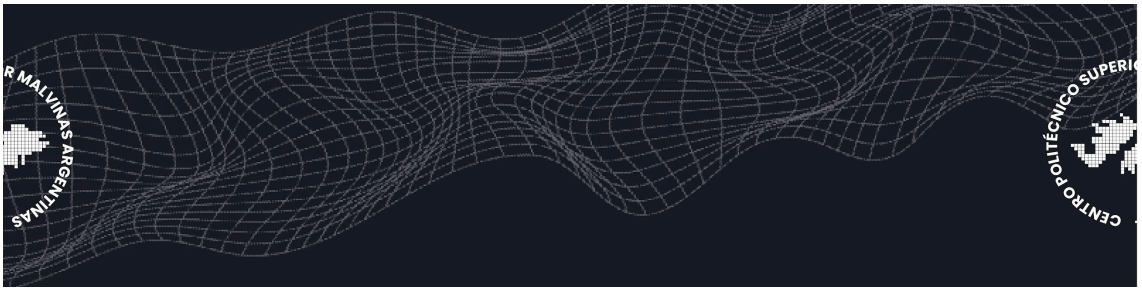
Arrays Multidimensionales

Los arrays multidimensionales son una generalización de los arrays bidimensionales y pueden tener más de dos dimensiones. Se utilizan en situaciones donde se necesita una estructura de datos con niveles de anidamiento más profundos.

Ejemplo de un array tridimensional en Python:

```
array_tridimensional = [
  [
    [1, 2, 3],
    [4, 5, 6]
  ],
  [
    [7, 8, 9],
    [10, 11, 12]
  ]
]
```

En este caso, el array tiene tres dimensiones: capas, filas y columnas. Puedes acceder a elementos específicos utilizando índices en cada dimensión.

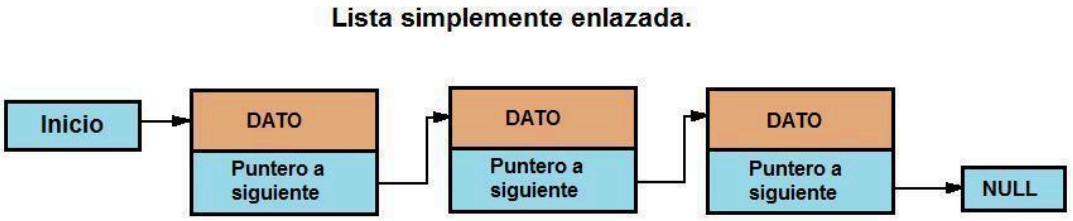


Los arrays de varias dimensiones son fundamentales en la programación cuando necesitas representar datos estructurados en una forma más compleja que un vector unidimensional. Son utilizados en una variedad de aplicaciones, como procesamiento de imágenes, simulaciones científicas, álgebra lineal y más, donde la estructura de datos debe ser multidimensional para reflejar la naturaleza de los datos que se manejan.

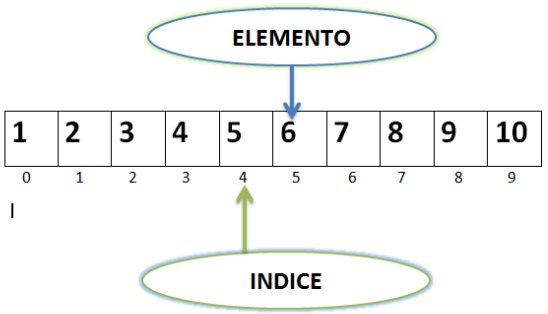
Colecciones – estructuras de datos dinámicas.

Las estructuras de datos dinámicas son tipos de estructuras utilizadas en programación que permiten la gestión flexible de datos, ya que pueden aumentar o reducir su tamaño durante la ejecución de un programa. Estas estructuras utilizan asignación de memoria dinámica, lo que significa que se adaptan según las necesidades cambiantes del programa. Algunas de las estructuras de datos dinámicas más comunes incluyen listas enlazadas, pilas, colas, árboles y grafos, así como las colecciones proporcionadas por lenguajes de programación modernos, como listas en Python o ArrayLists en Java. Las estructuras de datos dinámicas son particularmente útiles cuando no se conoce de antemano la cantidad exacta de datos que se manejarán o cuando se busca optimizar la eficiencia en la gestión de la

memoria. Su capacidad para cambiar de tamaño las hace flexibles y versátiles en una amplia gama de aplicaciones de programación.

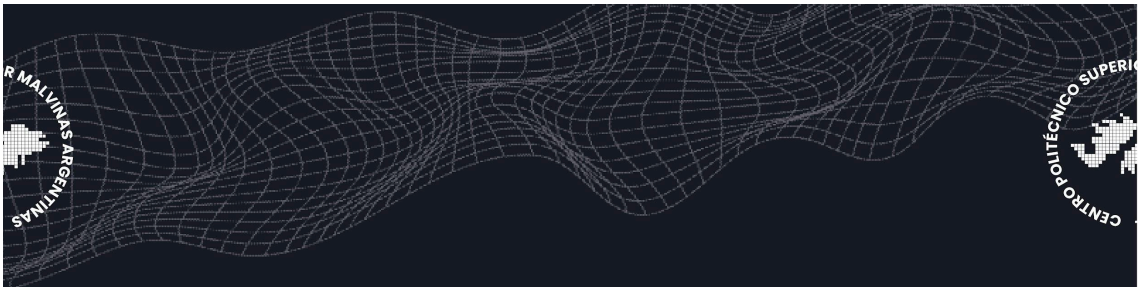


En el gráfico anterior podemos observar una lista enlazada simple, cada dato se ubica en cualquier ubicación de la memoria, a diferencia de un vector, que están todos los datos de forma contigua. Como lo muestra la siguiente imagen.



Las listas, tuplas, conjuntos y diccionarios

Listas (list)



Listas son colecciones ordenadas y mutables de elementos. Se definen utilizando corchetes [] y los elementos se separan por comas. Puedes agregar, eliminar y modificar elementos después de crear una lista.

Las listas son adecuadas cuando necesitas una colección ordenada y flexible de elementos.

```
mi_lista = [1, 2, 3]
mi_lista.append(4)  # Añadir un elemento
mi_lista.remove(2)  # Eliminar un elemento
```

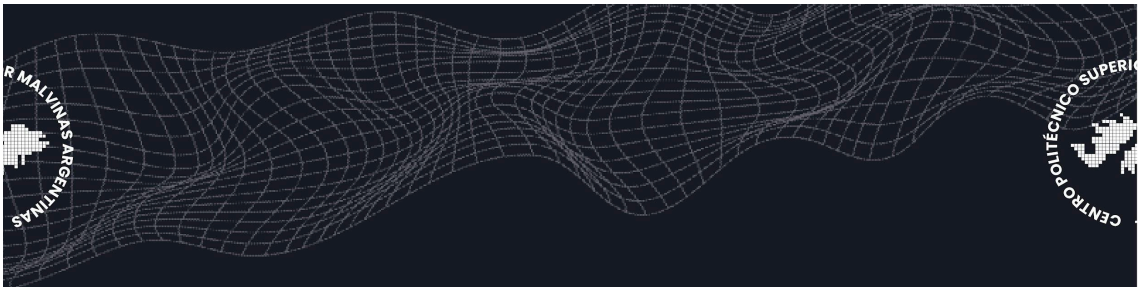
Nota importante

En Python, la mayoría de las estructuras de datos se gestionan de manera dinámica, lo que significa que no tienes que preocuparte por el tamaño o la gestión de la memoria de forma explícita. Sin embargo, es posible simular estructuras de datos estáticas utilizando algunas técnicas y bibliotecas específicas.

Tuplas (tuple)

Las tuplas son colecciones ordenadas e inmutables de elementos.

Se definen utilizando paréntesis () y los elementos se separan por comas.



Una vez creada una tupla, no puedes modificar sus elementos.

Son útiles cuando quieres datos que no deban cambiar, como coordenadas o claves de acceso.

```
mi_tupla = (1, 2, 3)
# Las tuplas son inmutables
```

Conjuntos (set)

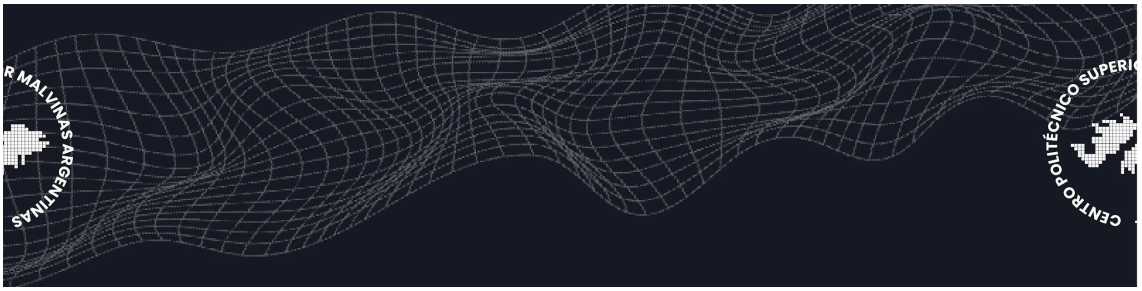
Los conjuntos son colecciones desordenadas y mutables de elementos únicos.

Se definen utilizando llaves {} o la función set(), y los elementos no tienen un orden específico.

No pueden contener elementos duplicados, lo que los hace ideales para eliminar duplicados de una lista.

Puedes agregar, eliminar y realizar operaciones de conjuntos, como intersección y unión.

```
mi_conjunto = {1, 2, 3}
mi_conjunto.add(4)  # Añadir un elemento
mi_conjunto.remove(2)  # Eliminar un elemento
```



Diccionarios (dict)

Los diccionarios son colecciones desordenadas de pares clave-valor.

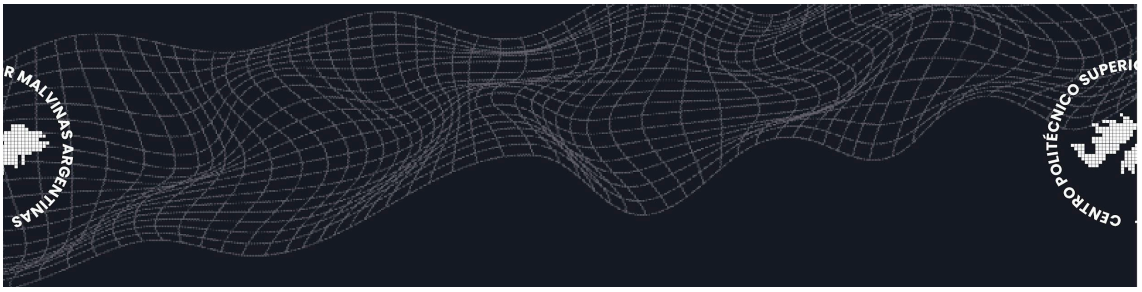
Se definen utilizando llaves {} y los pares clave-valor se separan por comas, con la clave y el valor separados por :.

Cada clave en un diccionario debe ser única.

Los diccionarios son útiles cuando necesitas asociar datos, como un diccionario inglés-español o información estructurada.

```
mi_diccionario = {'a': 1, 'b': 2}
mi_diccionario['c'] = 3  # Añadir un par clave-valor
del mi_diccionario['a']  # Eliminar un par clave-valor
```

Las *listas* son colecciones ordenadas y mutables, las *tuplas* son colecciones ordenadas e inmutables, los *conjuntos* son colecciones desordenadas de elementos únicos y los *diccionarios* son colecciones desordenadas de pares clave-valor. La elección de la estructura de datos depende de tus necesidades específicas en cuanto a orden, mutabilidad y asociación de datos en tu programa.

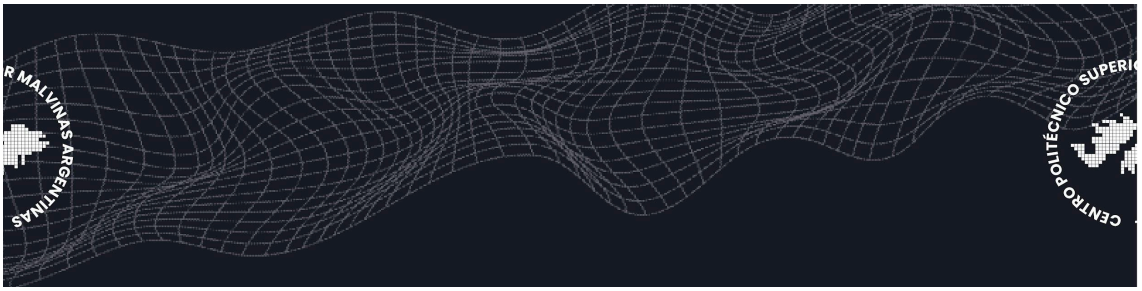


4- Actividad

a. Te proponemos tomar de ejemplo el juego TA-TE-TI, y que dividas en partes las acciones del juego como así también el orden en que se ejecutan. (aquí estás ejercitando la división de un problemas en partes, módulos)
b. ¿Qué estructura de datos utilizarías para representar el tablero?
c. ¿Te animas a programarlo?

5- Resumen

Estos conceptos forman una base sólida para comprender cómo organizar y manipular datos en programación, lo que es esencial para el desarrollo de aplicaciones y algoritmos efectivos.



Bibliografía

· FUNDAMENTOS DE PROGRAMACIÓN. Algoritmos, estructura de datos y objetos Cuarta edición - Luis Joyanes Aguilar - McGraw-Hill - 2008 - ISBN 978-84-481-6111-8 Bibliografía sugerida de la Unidad:

· Programación y Algoritmos - Aprenda a Programar en C y Pascal: Manuales Users - Maximiliano Bonanata - M.P. Ediciones - 2003 - ISBN 9875261564.

· La biblia del programador, Implementación y debugging: Manuales Users .code - Dante Cantone - M.P. Ediciones - 2003 - ISBN 9872299579