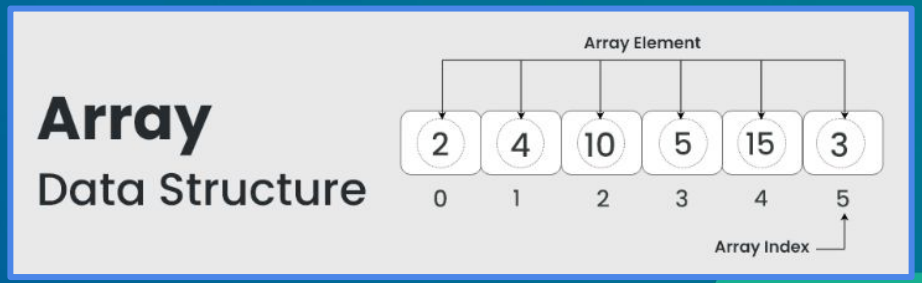
Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Arreglos unidimensionales: Vectores

¿Qué es un vector?

Un **arreglo unidimensional**, también conocido como **vector**, es una **estructura de datos que organiza elementos de manera lineal**, es decir, dispuestos uno tras otro en una sola dimensión. Cada elemento tiene una posición única y puede ser accedido utilizando un único índice.



Beneficios del uso de vectores/listas :

- **Organización y gestión de datos:** Las listas permiten agrupar datos relacionados en una sola estructura, simplificando su manejo y organización.
- **Acceso y manipulación eficiente:** Las listas facilitan el acceso, la iteración y la modificación de datos mediante índices, lo que es más eficiente que manejar variables individuales.
- **Escalabilidad:** Las listas son flexibles y escalables, permitiendo ajustar la cantidad de datos sin necesidad de crear nuevas variables.
- **Operaciones en lote:** Las listas permiten realizar operaciones en grupo, como ordenar y filtrar, de manera más eficiente que aplicar operaciones individuales a cada variable.

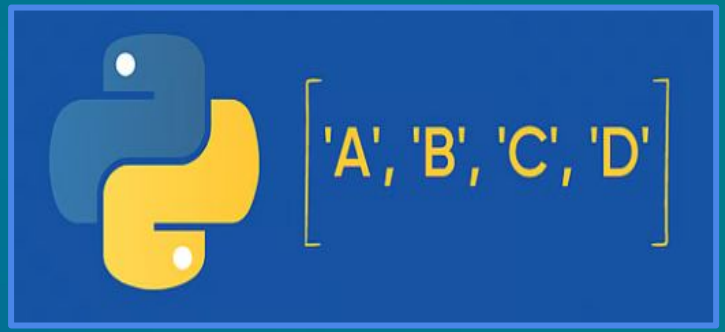
Arrays en Python

En Python, utilizaremos listas simples para representar arrays, enfocándonos en su uso algorítmico. Las listas almacenan colecciones de objetos de manera ordenada y accesible, siendo útiles para datos relacionados.

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Arreglos Unidimensionales

- **Las listas son estructuras de datos que almacenan elementos en un orden secuencial.**
- **Los elementos dentro de una lista pueden ser de diferentes tipos (enteros, cadenas, booleanos, etc.)**



Características de listas :

- **Mutable:** los elementos de una lista pueden cambiar luego de su creación.
- **Indexable:** los elementos de una lista pueden accederse mediante índices (posiciones).
- **Ordenada:** Los elementos de una lista mantienen un orden específico (no significa que los elementos están ordenados, por ejemplo de menor a mayor)

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Problema: gestión de calificaciones de estudiantes

Imaginá que necesitas gestionar las calificaciones de 100 estudiantes en una clase. Usar variables individuales para cada calificación sería impráctico y engorroso. Por ejemplo, si intentas hacerlo con variables individuales, tendrías que definir 100 variables, una para cada estudiante, y luego realizar operaciones en cada una de ellas.

Problema: gestión de calificaciones de estudiantes

Aquí, gestionar y operar con tantas variables individuales es extremadamente complicado y propenso a errores.

```
#Ejemplo impráctico con variables individuales
calificacion1 = 85
calificacion2 = 90
calificacion3 = 78
#...
calificacion100 = 88

#Imagina que necesitas calcular el promedio
promedio = (calificacion1 + calificacion2 + calificacion3 + ... + calificacion100) / 100
```

Solución con listas

Las listas permiten almacenar todas las calificaciones en una sola estructura, lo que facilita la gestión y manipulación de los datos.

```python
# Lista de 100 calificaciones (puedes reemplazar estos valores con los tuyos)
calificaciones = [85, 90, 78, 92, ..., 86]

# Inicializar la variable suma en 0
suma = 0

# Ciclo for para acceder a las calificaciones por su índice y sumarlas
for i in range(len(calificaciones)):
    suma += calificaciones[i]

# Calcular el promedio
promedio = suma / len(calificaciones)
```

Explicación

- **Organización:** Con una lista, podés almacenar todas las calificaciones en un solo lugar, en lugar de tener que definir 100 variables.
- **Acceso y Manipulación:** Podés acceder a cualquier calificación usando su índice y realizar operaciones en todas las calificaciones de manera eficiente.
- **Escalabilidad:** Si agregás más estudiantes, solo necesitas añadir elementos a la lista en lugar de crear nuevas variables.

Declaración de una lista vacía en Python

- **Declaración Básica:** Crea una lista vacía usando corchetes

```
lista_vacia = []
```

- **Uso del Constructor:** Alternativamente, usa el constructor ‘list()’ para crear una lista vacía

```
lista_vacia = list()
```

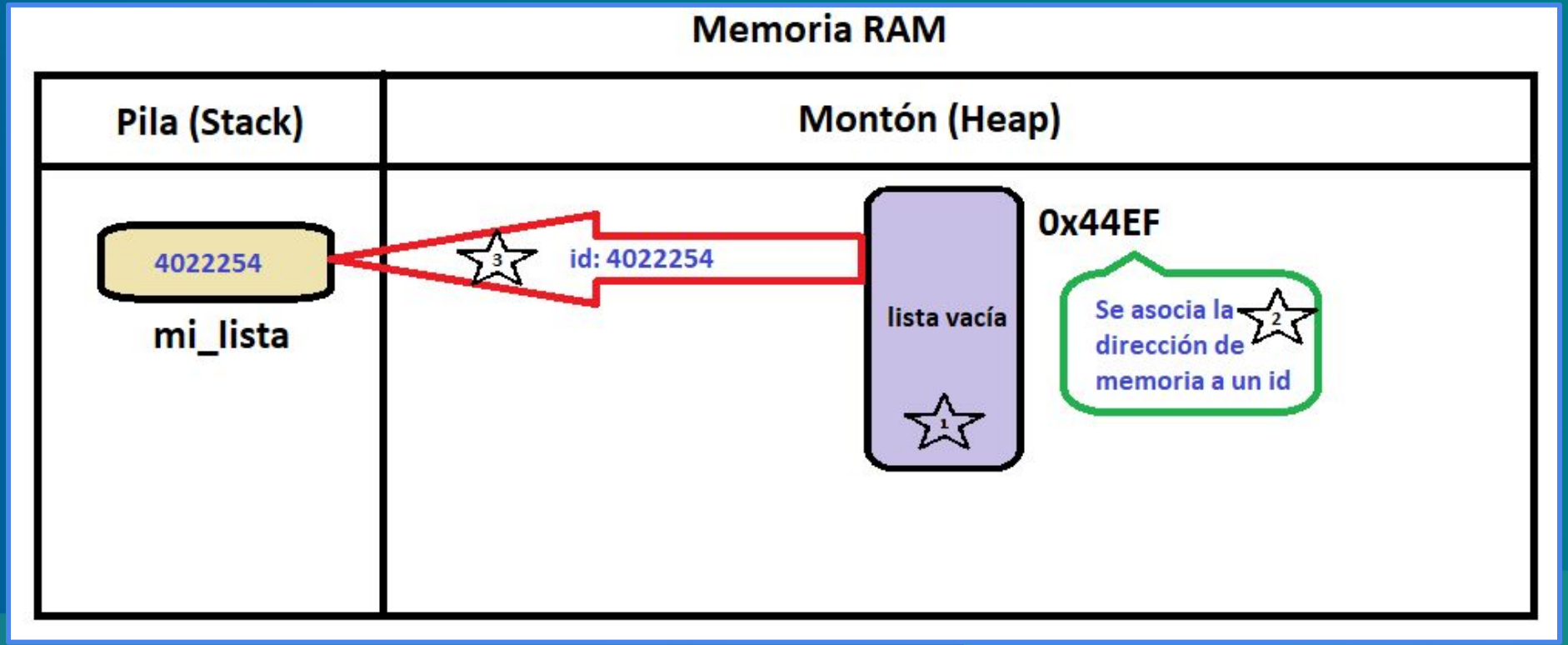
Comportamiento en la memoria - Stack y Heap

Estas instrucciones crean una lista vacía y asigna esa lista a la variable mi_lista.

¿Qué sucede en memoria?

1. Se reserva un espacio en memoria para almacenar la lista vacía. Este espacio contendrá la información necesaria para administrar la lista, como su tamaño, capacidad (espacio asignado en memoria) y una referencia a los elementos de la lista.
2. Se asigna un identificador único a esta lista en memoria (su id)
3. Se asocia la variable con este espacio de memoria que contiene la lista vacía en memoria.

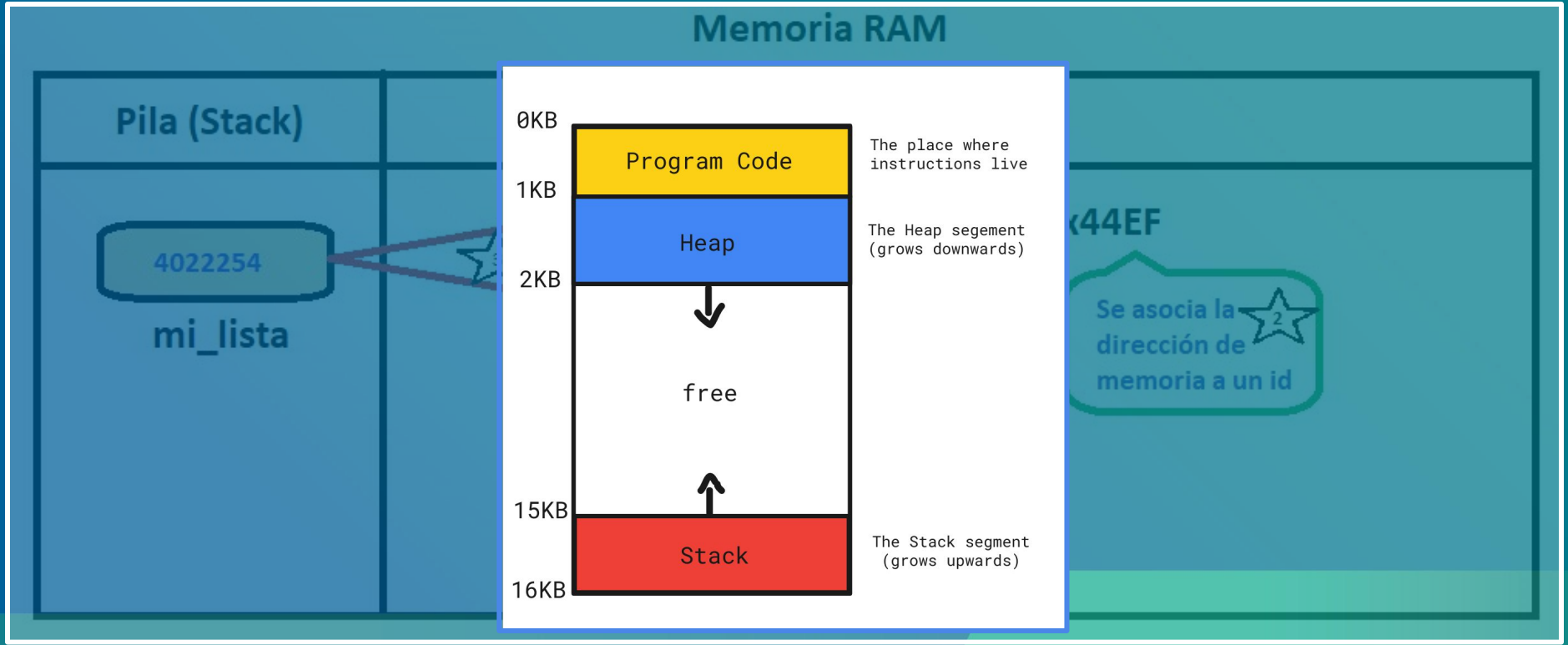
Arreglos Unidimensionales



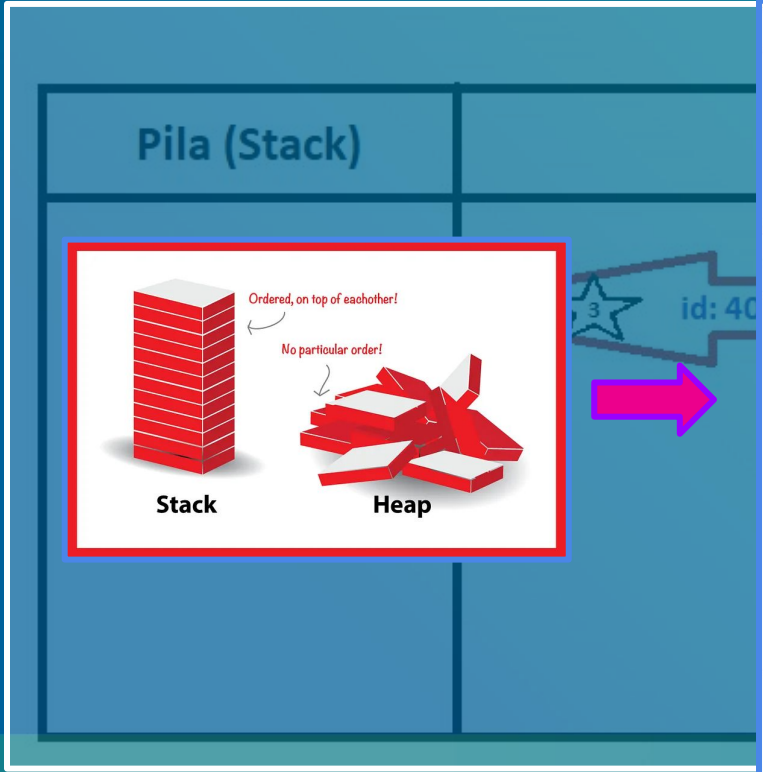
Programación I

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Arreglos Unidimensionales



Arreglos Unidimensionales



	Stack (Pila)	Heap (Montón)
Asignación	*Estática: es una región de memoria de tamaño fijo y predefinido (por lo general muy pequeño respecto de los otros segmentos) que se asigna en tiempo de ejecución.*	*Dinámica: es una región de memoria más grande que la pila y dinámica que se asigna cuando se solicita durante la ejecución del programa. Es decir, las variables pueden cambiar su tamaño. Por ejemplo una lista.*
Almacenamiento	*Variables locales: se utiliza para almacenar valores locales y retornos de una función.*	*Almacenamiento de objetos, listas, diccionarios*
Gestión	*Es gestionada de manera automática en tiempo de ejecución*	*Es gestionada de manera manual o automática (dependiendo del lenguaje).* ***Manual****: lo hace el programador. El mismo es encargado de reservar y liberar la memoria para una variable determinada. Por ejemplo en C.* ***Automática****: lo gestiona el lenguaje. El mismo reserva espacio en memoria dinámica y al momento de liberar la misma se apoya en un software denominado Recolector de Basura (Garbage Collector). Por ejemplo C#, Phyton, etc.*
Acceso	*Funcionamiento LIFO (último entrado, primero salido)*	*El acceso no es secuencial. No sigue un orden específico.*
	Las operaciones de heap pueden ser más lentas que en el stack debido a la necesidad de asignar y liberar memoria de manera dinámica.	

Inicialización de una lista

Definición de una lista utilizando corchetes:

```
nombres = ['Rosita', 'Manuel', 'Luciana']
nums = [92, 38, 42, 65, 11]
```

Inicializar a 0 una lista de 5 elementos:

```
#Inicializar una lista de 5 elementos con valores 0
mi_lista = [0] * 5

#Ciclo for para asignar a cada posición de la lista el valor de su íncice
for i in range(len(mi_lista)):
    mi_lista[i] = i
```

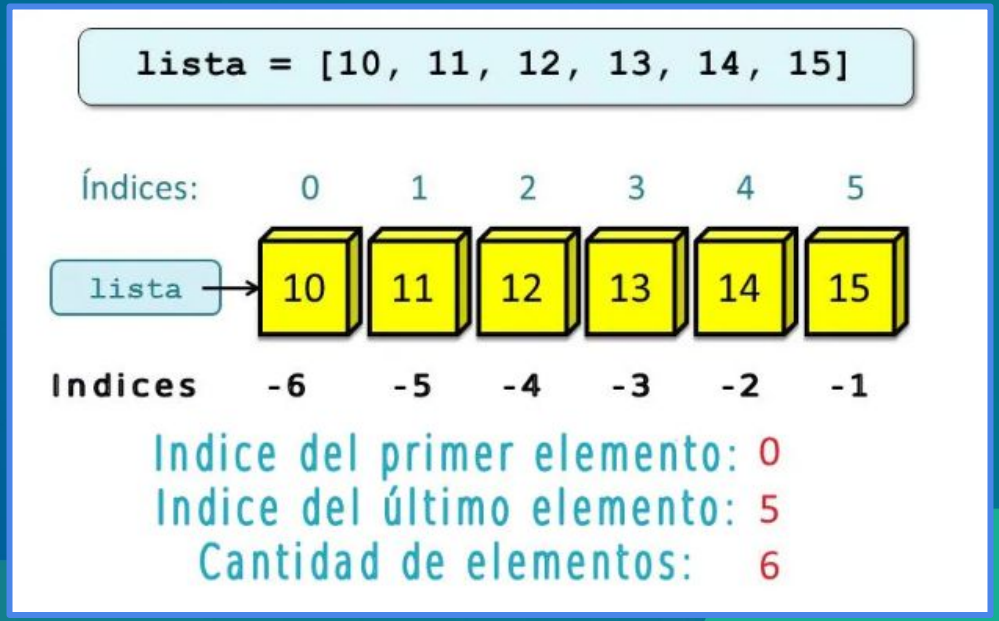
Manipulación de elementos usando índices

- Las listas se indexan de forma automática, donde el primer elemento está en la posición 0, el segundo en la posición 1, y así sucesivamente

```
#Creación de la lista

lista = [10, 11, 12, 13, 14, 15]
```

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Manipulación de elementos usando índices



Manipulación de elementos usando índices

```python
mi_lista = [10, "hola", True, 3.5]

# Acceder al primer elemento
print(mi_lista[0])  # Output: 10

# Acceder al segundo elemento
print(mi_lista[1])  # Output: hola

# Acceso con índice negativo (último elemento)
print(mi_lista[-1])  # Output: 3.5

```

Modificación de elementos por índice

Modificar un elemento a través del índice :

```
1   mi_lista = [10, "hola", True, 3.5]
2
3   mi_lista[1] = "adiós"
4   print(mi_lista)  # Output: [10, "adiós", True, 3.5]
5
```

Función len()

- Las listas en Python son dinámicas, lo que significa que puedes agregar o eliminar elementos durante la ejecución.
- Para verificar la longitud (cantidad de elementos) actual de una lista, se utiliza la función len().
- La función len() toma el nombre de la lista como argumento y devuelve un número entero representando la cantidad de elementos dentro de la lista.

Función len()

```python
# Definir una lista
lista = [10, 11, 12, 13, 14, 15]

# Obtener la longitud de la lista
longitud = len(lista)

# Imprimir la longitud de la lista
print("La longitud de la lista es:", longitud)
```

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Mostrar una lista

```
mi_lista = [4, 9, 3, 7, 1]

print(mi_lista)
```

```
mi_lista = [4, 9, 3, 7, 1]

for i in range(len(mi_lista)):
    print(mi_lista[i])
```

Usando el bucle for para procesar listas

- El bucle for en Python es una herramienta efectiva para recorrer y procesar listas.
- Para calcular la suma de todos los elementos de una lista, se puede usar una variable acumuladora (en este caso total) y un bucle que recorra los elementos de la lista.

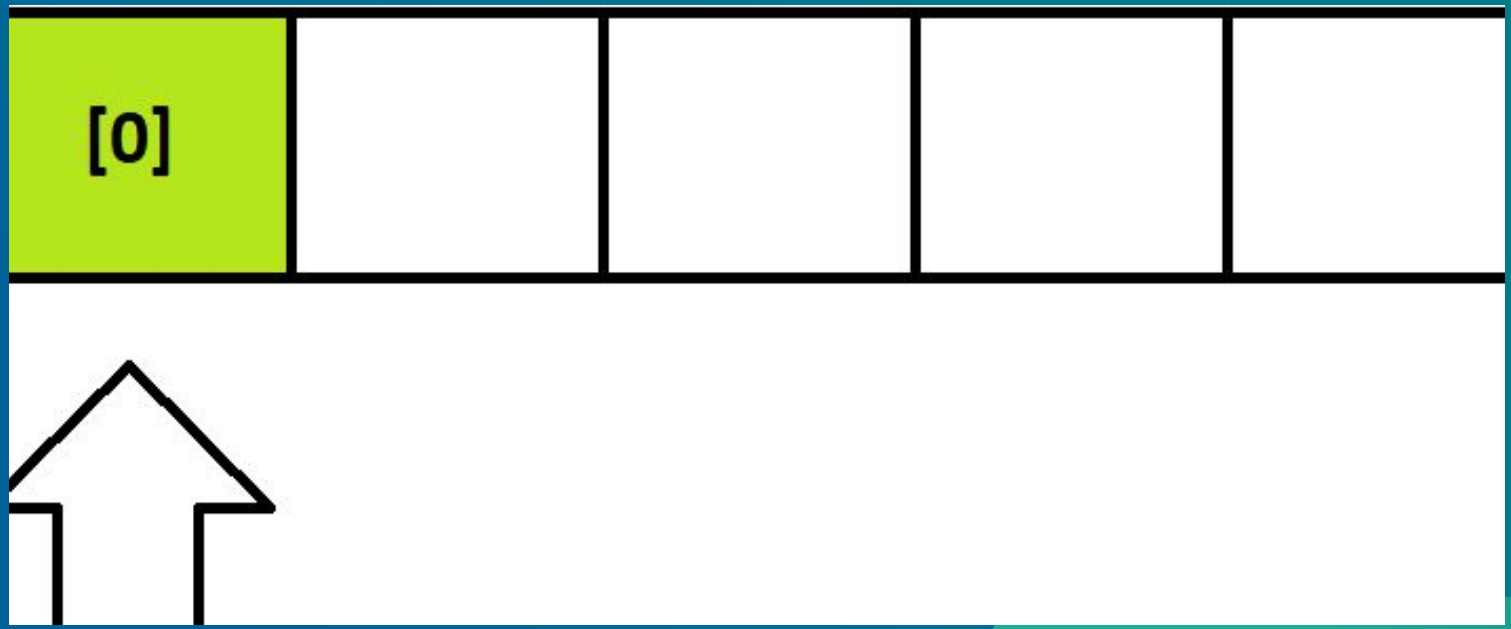
Nota: Es una buena práctica evitar el uso de nombres como sum para variables, ya que esto puede entrar en conflicto con funciones integradas como sum().

Usando el bucle for para procesar listas

```python
#Lista con 5 valores enteros
my_list = [1, 2, 3, 4, 5]

#Variable acumuladora para la suma
total = 0

#Usar un bucle for para recorrer la lista y sumar sus elementos
for i in range(len(my_list)):
    total += my_list[i]

#Imprimir el resultado de la suma
print("La suma total es:", total )
```

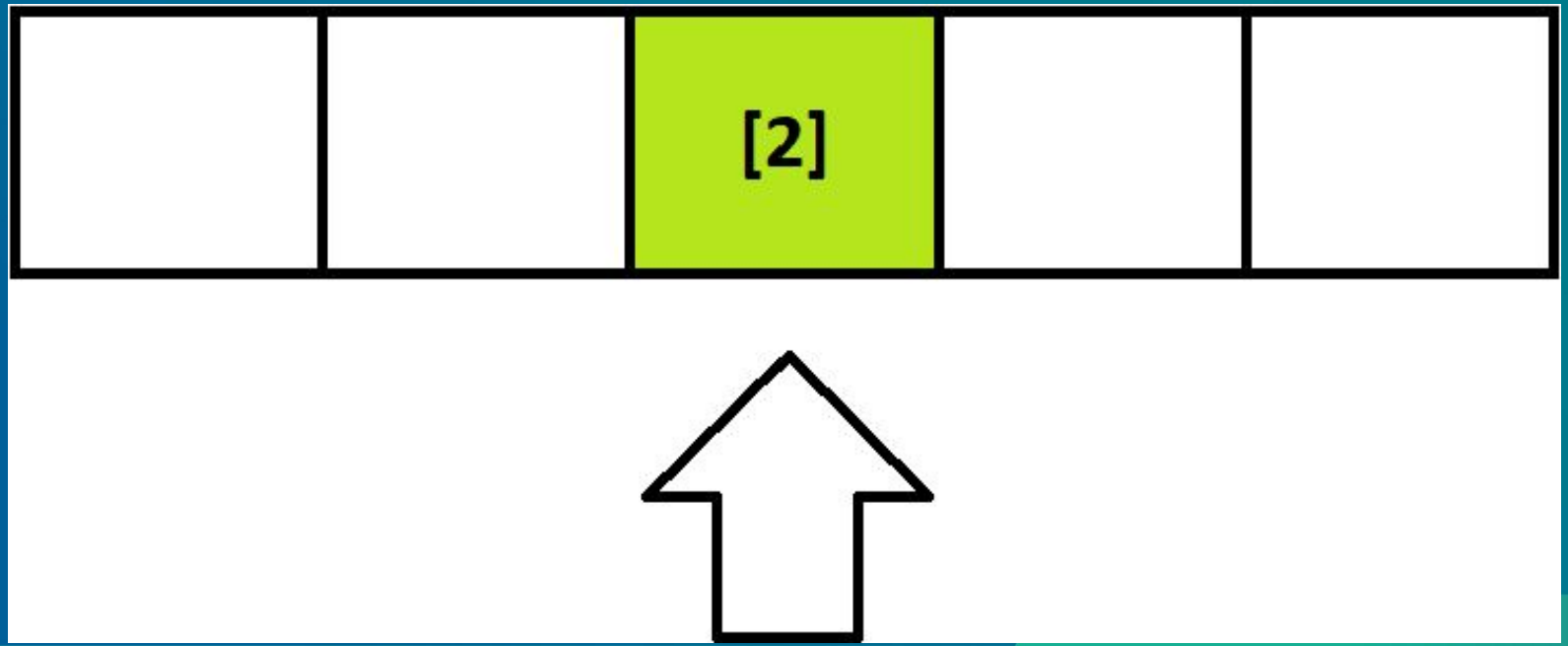
Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Cargar una lista de forma secuencial



Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Cargar una lista con acceso aleatorio



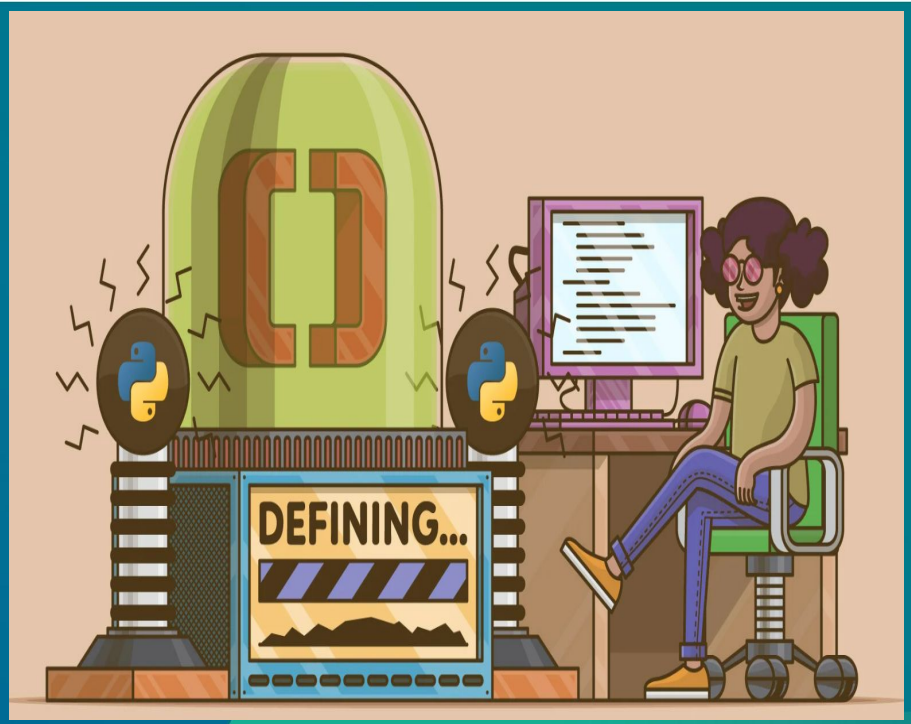
Cargar una lista con acceso aleatorio

Definir manualmente la posición y valor para cargar en una lista.

- El usuario puede definir tanto el valor que desea agregar a la lista como la posición en la que desea insertarlo.
- Esto proporciona al usuario control total sobre la estructura y contenido de la lista.

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Operaciones en listas



Características clave de las listas

Las listas se comportan de manera diferente a las variables ordinarias (escalares).

Ejemplo sorprendente:

- Crea una lista de un elemento llamada list_1.
- Asigna list_1 a una nueva lista llamada list_2.
- Cambia el único elemento de list_1.
- Imprime list_2.

Resultado inesperado: list_2 mostrará [2], no [1].

Razón: Las listas y otras entidades complejas se almacenan de manera diferente que las variables ordinarias.

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Características clave de las listas

```python
#Crea una lista de un elemento llamada list_1
list_1 = [1]

#Asignar list_1 a una nueva lista llamada list_2
list_2 = list_1

#Cambia el único elemento de list_1
list_1[0] = 2

#Imprime la list_2
print(list_2) #Esto mostrará [2], no [1]
```

Entender la asignación en listas

Cómo funciona la asignación:

Variables ordinarias:

- El nombre de la variable es el nombre de su contenido.

Listas:

- El nombre de una lista es el nombre de una ubicación de memoria (su id) donde se almacena la lista.

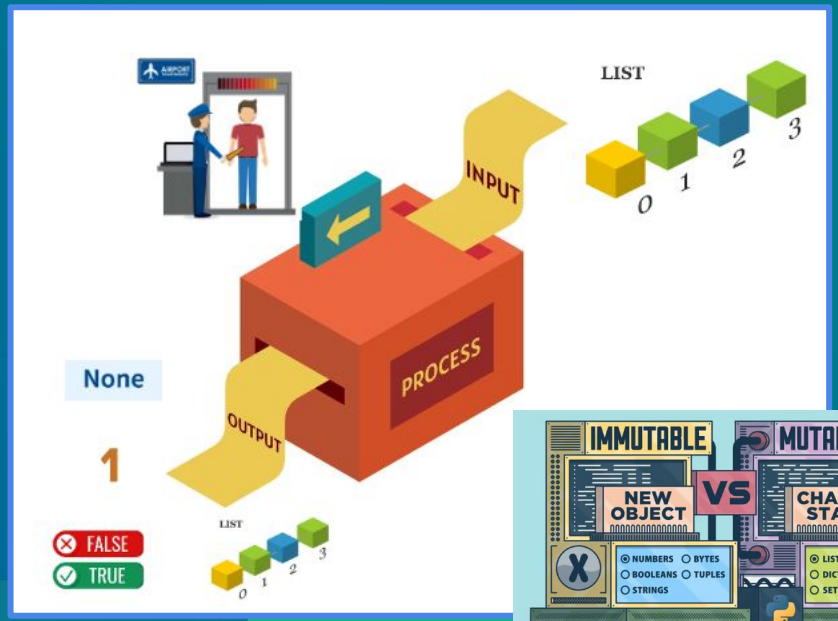
Entender la asignación en listas

- **Ejemplo clave:**
  - lista_2 = lista_1 copia el nombre del arreglo, no su contenido.
  - Ambos nombres (lista_1 y lista_2) identifican la misma ubicación en la memoria.
- **Implicación: modificar uno afecta al otro.**

Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Listas y funciones

- Las listas son objetos parametrizables, esto significa que es posible pasarlas como parámetros de una función.
- Siempre es recomendable realizar un control de tipo sobre el parámetro, es decir verificar si el type del parámetro es del tipo list.
- Según su procesamiento, estas funciones podrán retorna todos los tipos conocidos, incluyendo otras listas.
- No olvidar el carácter mutable de las listas.



Scudero Yanina - Scarafilo Germán - Costello Patricio - Benitez Daniel

Algoritmos de búsqueda

Según la naturaleza de la búsqueda, es posible implementar distintos algoritmos sobre las listas.