Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

Clase grabada

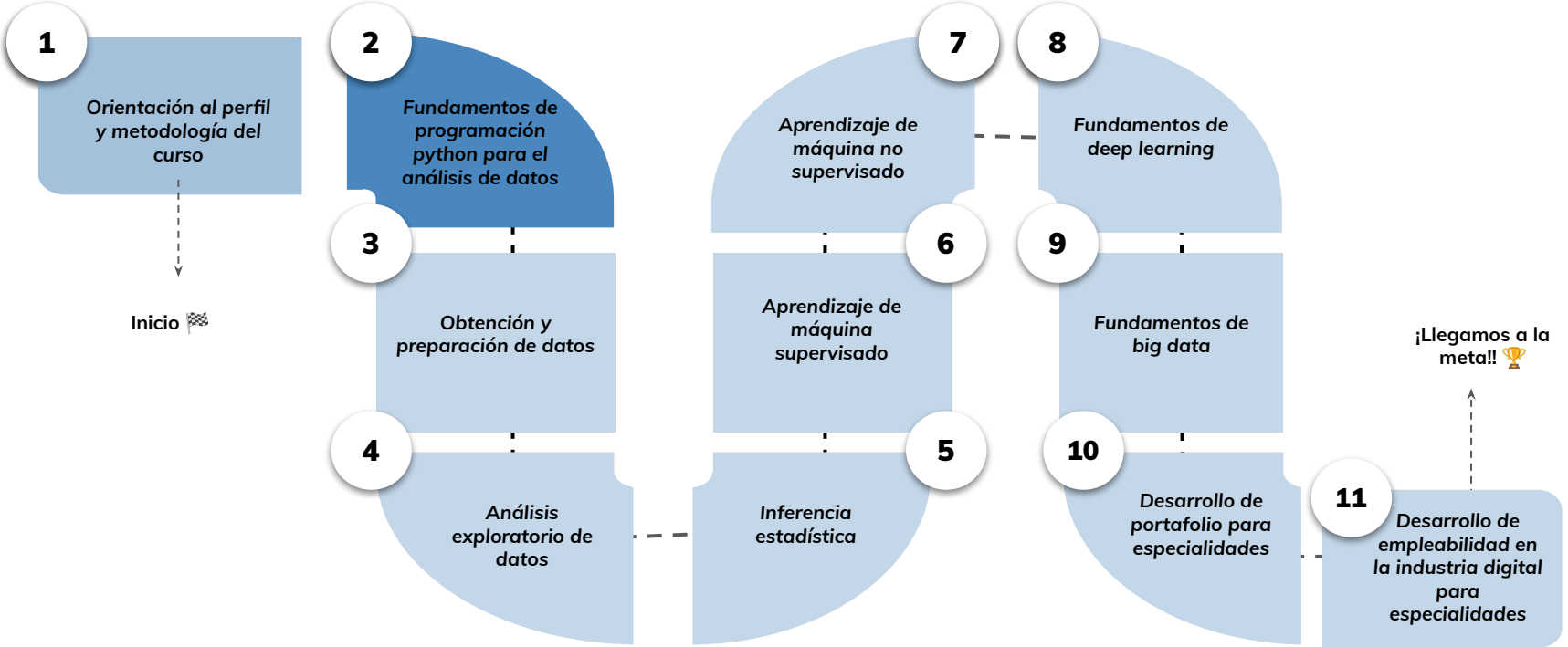
Estructuras de dato en Python - Parte I

Aprendizaje Esperado 5: Aplicar las estructuras de dato de tipo colección del lenguaje python junto a sus características y utilidad para resolver un problema.

alkemy

Hoja de ruta

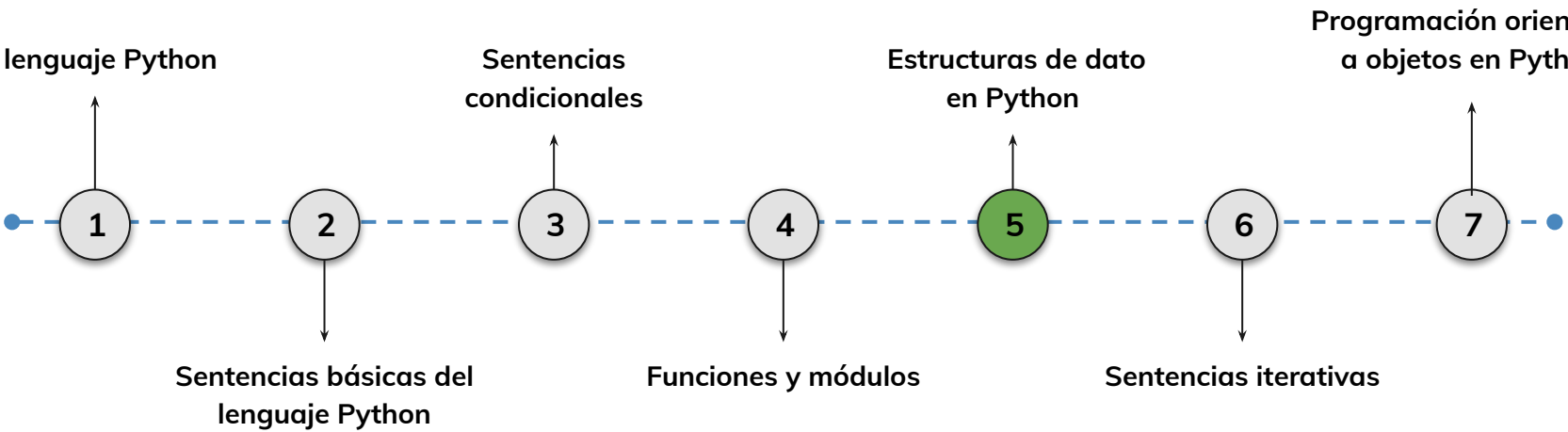
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?

5.

Uso práctico de estructuras de datos del lenguaje Python.

Veremos cuándo usar listas, diccionarios, tuplas y sets, sus operaciones más comunes y cómo elegir la mejor opción según el tipo de problema a resolver.

- **Listas**
  - definición, creación
  - acceso, métodos (append, pop, sort, etc.)
- **Diccionarios**
  - pares clave-valor
  - métodos (get, keys, items, etc.)

alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Identificar las estructuras de dato principales de Python: listas, diccionarios, tuplas y sets
- Comprender las características de cada estructura y su utilidad
- Manipular listas y diccionarios mediante sus métodos más frecuentes
- Acceder, modificar y eliminar elementos en estructuras mutables
- Aplicar estructuras de datos para resolver un problema de baja complejidad

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Aprendimos a crear y reutilizar funciones personalizadas*
- → *Usamos funciones preconstruidas y del módulo functools (reduce)*
- → *Modularizamos código para hacerlo más legible y reutilizable*
- → *Resolvimos un caso práctico de stock e inventario usando programación funcional*

Estructuras de dato en python

Introducción

Las estructuras de datos son esenciales en la programación, ya que permiten organizar y manipular grandes cantidades de información de manera ordenada y eficiente.

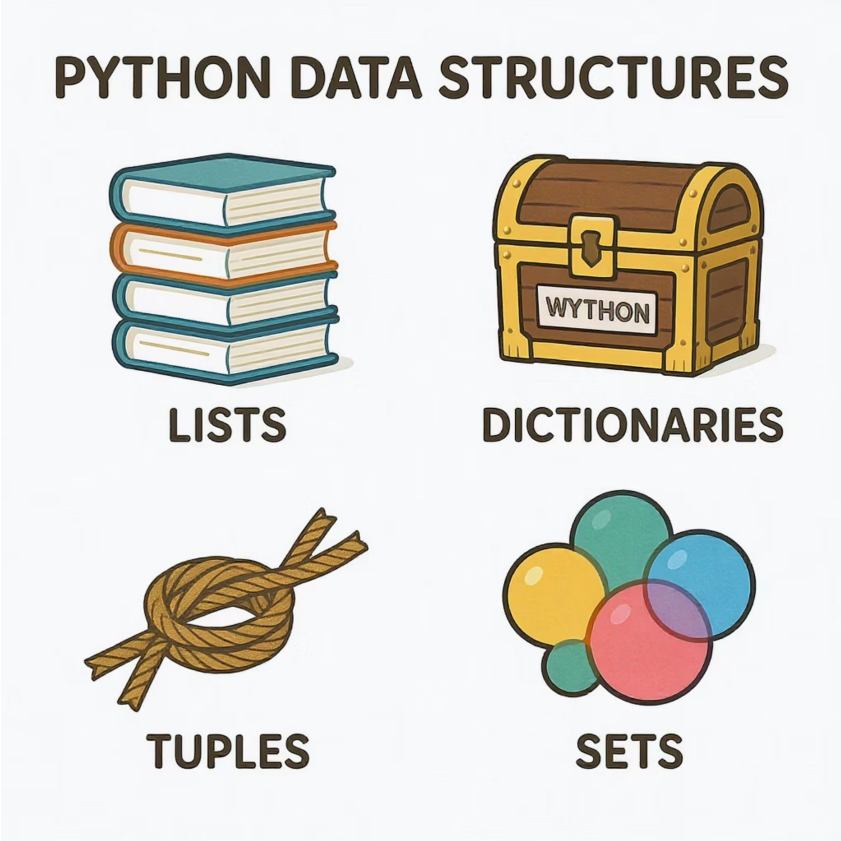
En Python, existen varias estructuras de datos integradas, cada una diseñada para almacenar datos de una forma específica, optimizando así el rendimiento y facilitando la realización de operaciones complejas. El conocimiento y uso correcto de estas estructuras permiten que los programadores desarrollen aplicaciones eficientes y bien estructuradas, desde proyectos simples hasta sistemas complejos.

alkemy

< >

Estructuras de datos en Python

Esta clase cubre las estructuras de datos más utilizadas en Python: listas, diccionarios, tuplas y sets. Cada una de estas estructuras tiene características únicas que determinan cuándo y cómo utilizarlas en un proyecto.



alkemy

Principales estructuras de datos

Listas

Permiten almacenar colecciones de elementos ordenados y modificables

Diccionarios

Ofrecen una estructura de clave-valor útil para búsquedas rápidas

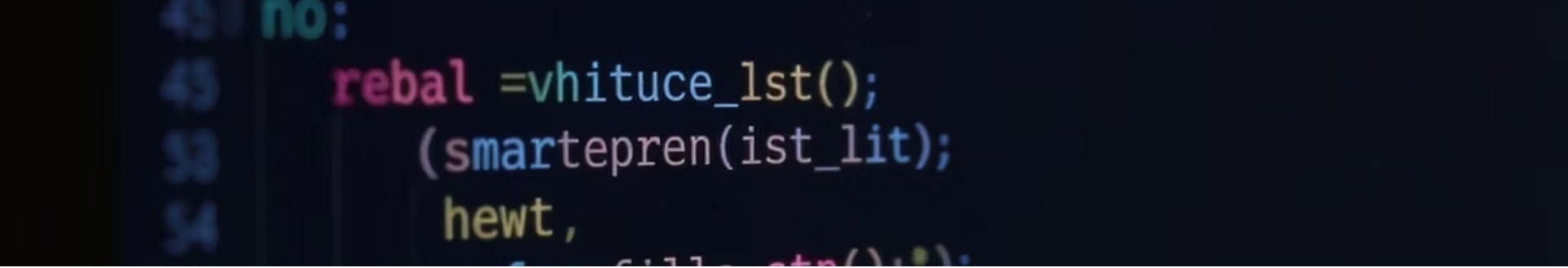
Tuplas

Almacenan datos inmutables, ideales para colecciones constantes

Sets

Eliminan elementos duplicados y permiten realizar operaciones de conjunto

alkemy



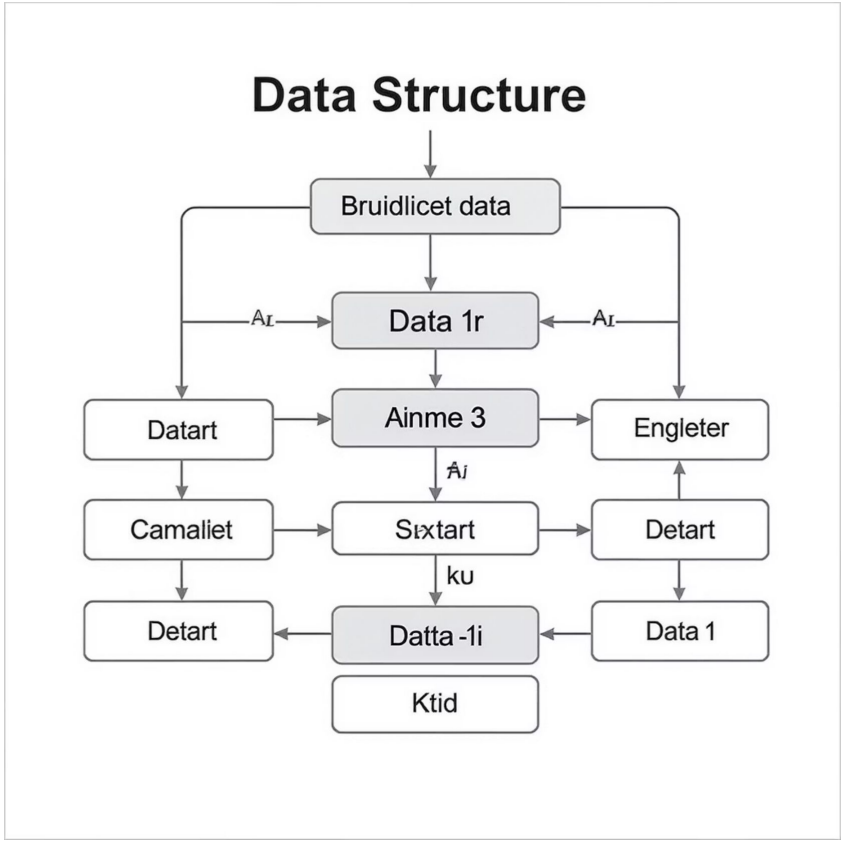
Compresión de estructuras

Además, se explorará el concepto de compresión de listas, diccionarios y sets, una técnica avanzada que permite escribir código más compacto y optimizado.

< >

¿Qué es una Estructura de Datos?

Una estructura de datos es una forma organizada de almacenar y gestionar datos en la memoria de un programa, lo que facilita su acceso y manipulación. Las estructuras de datos ayudan a organizar los datos de manera eficiente, permitiendo realizar operaciones complejas de forma rápida y ordenada.

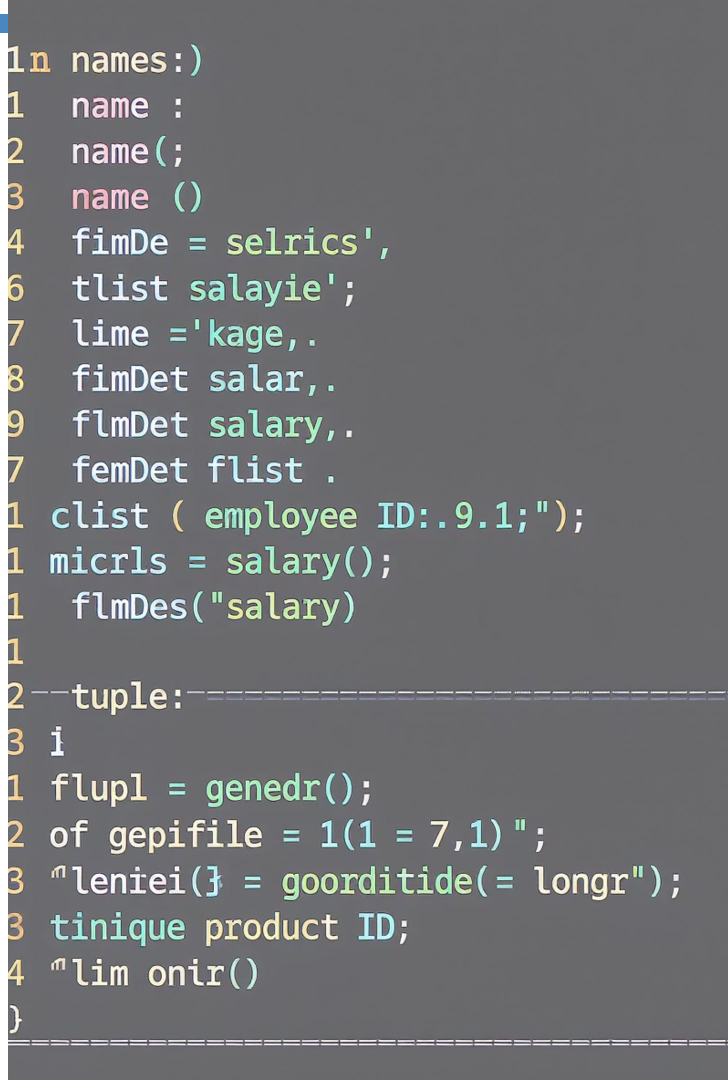


alkemy

< >

Estructuras principales en Python

En Python, las estructuras de datos principales incluyen listas, diccionarios, tuplas y sets, cada una adecuada para diferentes tipos de tareas y necesidades.



Importancia de la elección

La elección de una estructura de datos adecuada es crucial para el rendimiento y la eficiencia del programa, ya que cada estructura permite realizar diferentes tipos de operaciones con distinta rapidez. Por ejemplo, una lista permite acceder a elementos mediante índices, mientras que un diccionario permite asociar cada valor con una clave específica, facilitando la búsqueda y recuperación de datos.

< >

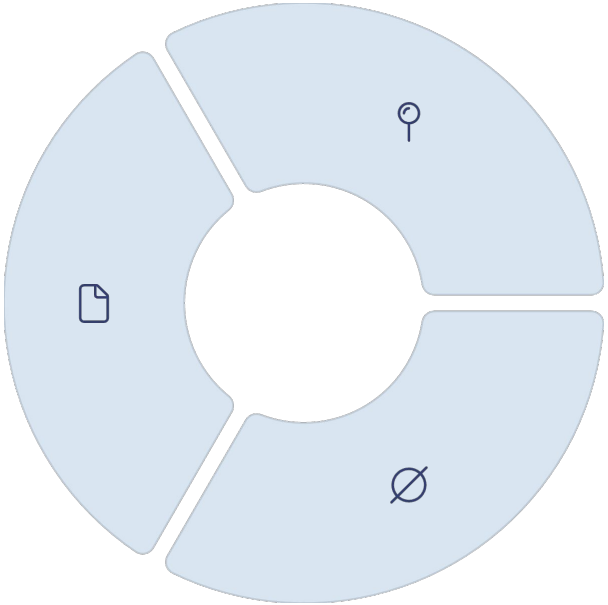
¿Por qué son importantes las estructuras de datos?

- Las estructuras de datos son esenciales para organizar y procesar grandes volúmenes de información de manera eficiente.
- Al elegir la estructura adecuada, el programa puede mejorar su rendimiento y optimizar el uso de la memoria.
- Esto es crucial en aplicaciones que manejan muchos datos o que requieren alta velocidad de procesamiento.

Clasificación de las Estructuras de Datos

Secuencias

Listas y tuplas



Mapeos

Diccionarios

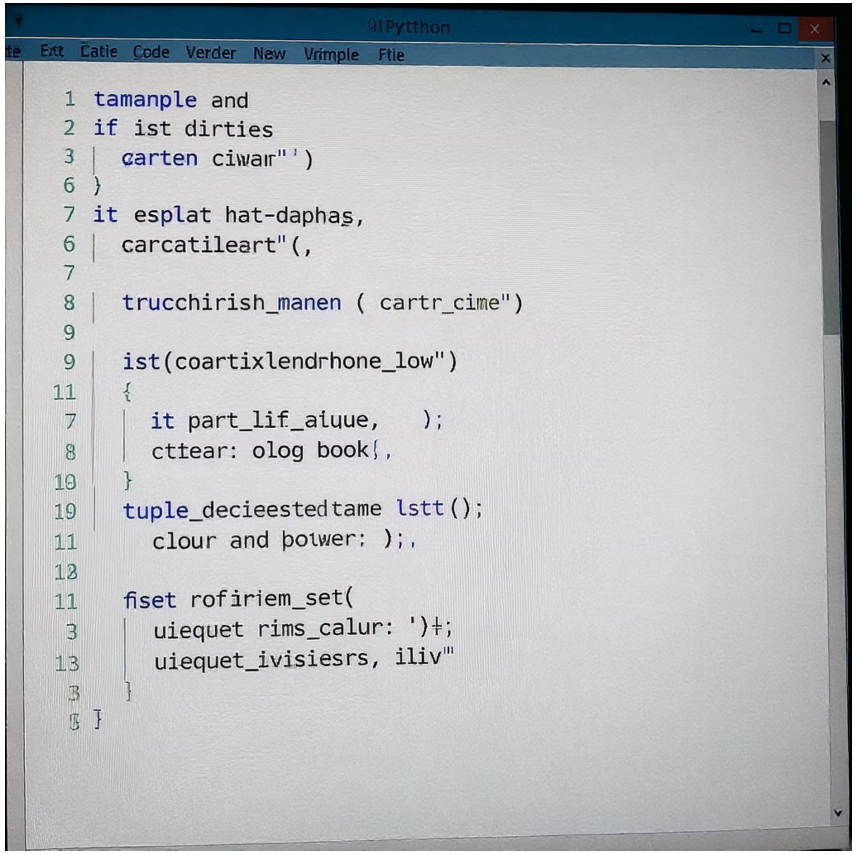
Conjuntos

Sets

Cada tipo ofrece operaciones específicas y es adecuado para resolver problemas concretos, lo cual permite a los programadores seleccionar la estructura que mejor se adapta a sus necesidades.

Ejemplos comunes de estructuras de datos

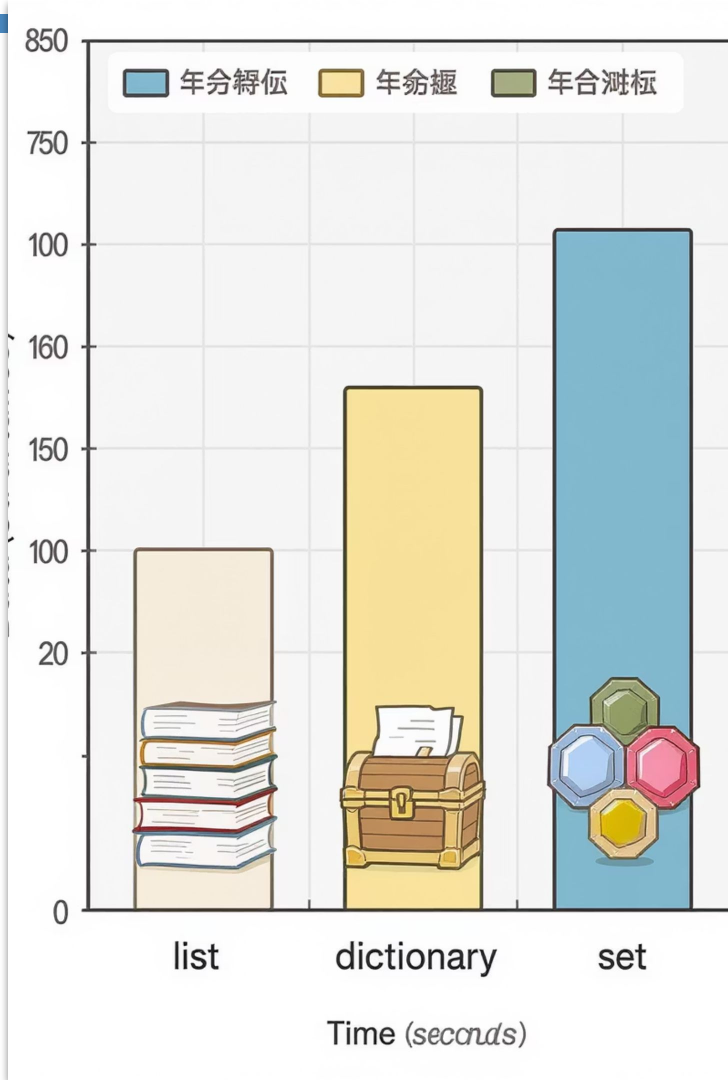
- Una lista puede almacenar una secuencia ordenada de elementos
- Un diccionario es ideal para almacenar pares clave-valor
- Un set es útil para eliminar duplicados y realizar operaciones de conjunto



alkemy

Importancia de elegir la estructura adecuada

Cada estructura de datos tiene sus ventajas y limitaciones. La elección incorrecta puede llevar a un uso ineficiente de la memoria o a un rendimiento lento. Por ello, es importante comprender las características de cada estructura para tomar decisiones informadas que mejoren la eficiencia y la claridad del programa.



Listas en Python

Características de una Lista

Las listas en Python son estructuras de datos ordenadas y mutables que permiten almacenar múltiples elementos en una secuencia. A diferencia de las tuplas, las listas son dinámicas, lo que significa que sus elementos se pueden modificar, agregar o eliminar después de su creación. Las listas son útiles para manejar colecciones de datos que cambian durante la ejecución del programa.

Versatilidad de las listas

Una lista en Python puede contener diferentes tipos de datos, como enteros, cadenas, e incluso otras listas, lo cual permite almacenar información compleja en una sola estructura.

Crear una Lista

Para crear una lista en Python, se utilizan corchetes [] y los elementos se separan por comas. Una lista puede contener cualquier número de elementos, incluyendo una lista vacía, y los elementos no necesitan ser del mismo tipo.

```
mi_lista = [1, 2, 3, "texto", 4.5]
print(mi_lista)  # Salida: [1, 2, 3, "texto", 4.5]
```

Agregar Elementos a una Lista

Python ofrece métodos como append() para agregar elementos al final de la lista y insert() para añadir elementos en una posición específica.

```
mi_lista = [1, 2, 3]
mi_lista.append(4)       # Agrega el 4 al final de la lista
mi_lista.insert(1, "a")  # Inserta "a" en la posición 1
print(mi_lista)          # Salida: [1, 'a', 2, 3, 4]
```

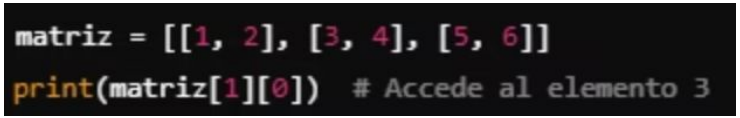
Rescatar un Elemento y Rango de Elementos

Para acceder a un elemento específico en una lista, se utiliza su índice, que comienza en 0. Además, se pueden extraer sublistas utilizando el slicing.

```
mi_lista = [1, 2, 3, 4, 5]
print(mi_lista[2])      # Salida: 3
print(mi_lista[1:4])    # Salida: [2, 3, 4]
```

Listas Anidadas y Matrices

Una lista puede contener otras listas como elementos, lo que permite crear estructuras complejas como matrices y listas de listas. Esto es útil para representar datos tabulares o jerárquicos.



Operaciones comunes con listas

Ordenar elementos

Usando el método sort() o la función sorted()

Eliminar elementos

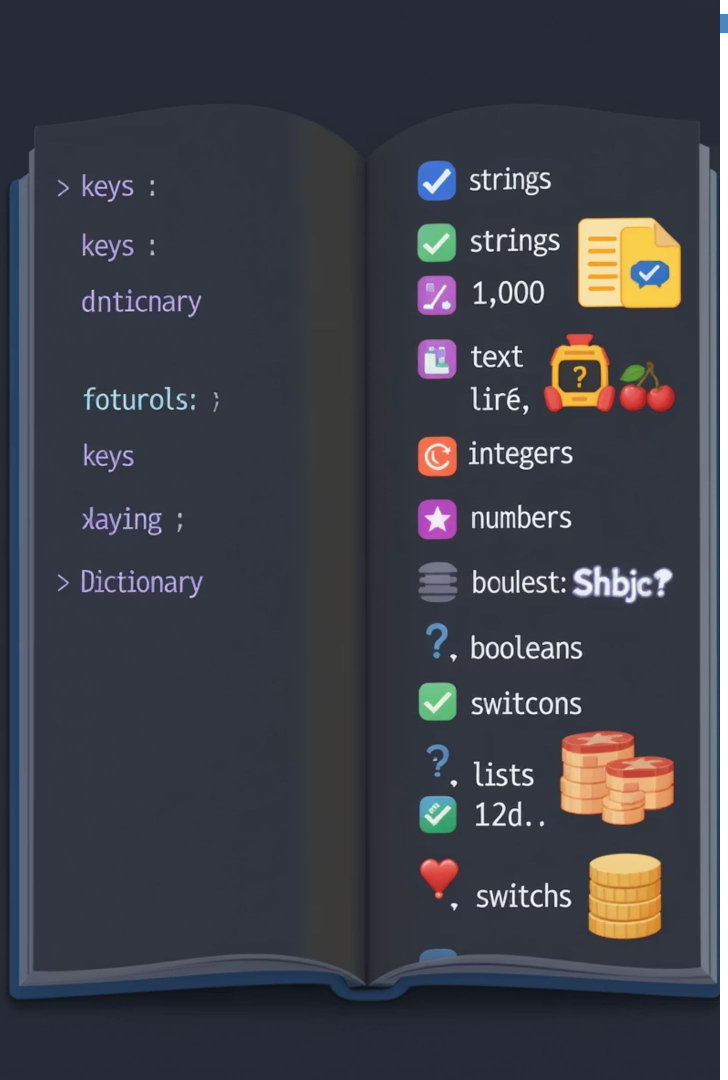
Con remove(), pop() o del

Encontrar elementos

Usando index() o in para verificar existencia

Combinar listas

Con el operador + o el método extend()



Diccionarios en Python

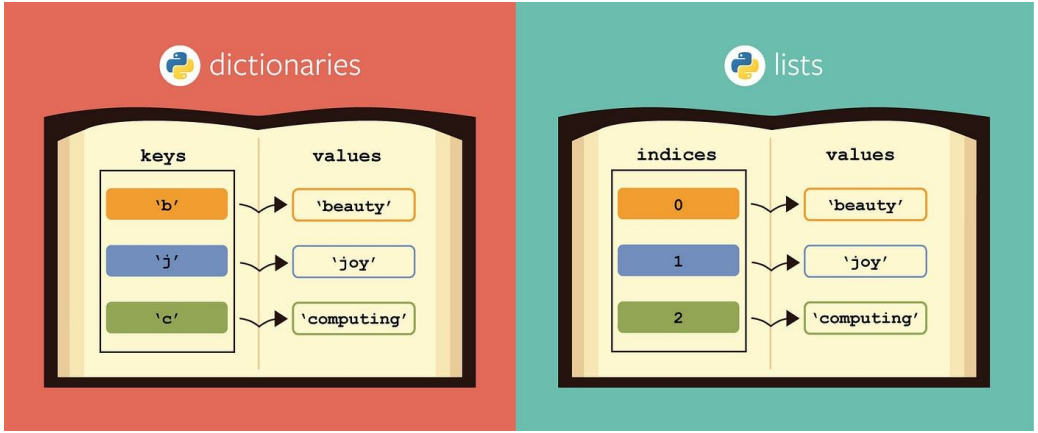
Características de un Diccionario

Los diccionarios son estructuras de datos que almacenan pares clave-valor, donde cada clave es única y se asocia con un valor específico. A diferencia de las listas, los elementos de un diccionario no están ordenados y se accede a ellos mediante sus claves, no mediante un índice.

alkemy

Ventajas de los diccionarios

Los diccionarios son mutables, lo que significa que se pueden modificar después de su creación, y son ideales para almacenar datos estructurados y realizar búsquedas rápidas.



< >

Crear un Diccionario

Para crear un diccionario en Python, se utilizan llaves {} y los pares clave-valor se separan por dos puntos :. Las claves pueden ser de cualquier tipo inmutable, como cadenas o números.

```
mi_diccionario = {"nombre": "Ana", "edad": 25, "ciudad": "Lima"}
print(mi_diccionario)  # Salida: {'nombre': 'Ana', 'edad': 25, 'ciudad': 'Lima'}
```

Agregar y Modificar elementos en un Diccionario

Para agregar o modificar elementos en un diccionario, se asigna un valor a una clave. Si la clave ya existe, se actualiza su valor; si no existe, se añade un nuevo par clave-valor.

```
mi_diccionario["profesion"] = "Ingeniera"
mi_diccionario["edad"] = 26
print(mi_diccionario)  # Salida: {'nombre': 'Ana', 'edad': 26, 'ciudad': 'Lima'
```

Rescatar Elementos en un Diccionario

Para acceder a un valor en un diccionario, se utiliza su clave entre corchetes o el método get() que devuelve None si la clave no existe, en lugar de generar un error.

```
print(mi_diccionario["nombre"])  # Salida: Ana
print(mi_diccionario.get("pais", "No especificado"))  # Salida: No especificado
```

< >

Diccionarios Anidados

Los diccionarios pueden contener otros diccionarios como valores, lo que permite crear estructuras de datos complejas y jerárquicas.

```
empleado = {
    "nombre": "Carlos",
    "datos_personales": {
        "edad": 35,
        "ciudad": "Bogotá"
    },
    "salario": 5000
}
print(empleado["datos_personales"]["ciudad"])  # Salida: Bogotá
```

< >

Operaciones comunes con diccionarios

- **Obtener todas las claves**
  Con el método keys()
- **Obtener todos los valores**
  Con el método values()
- **Obtener pares clave-valor**
  Con el método items()
- **Eliminar elementos**
  Con pop(), popitem() o del

Casos de uso de diccionarios

Almacenamiento de datos

Ideal para guardar información estructurada como perfiles de usuario

Búsquedas rápidas

Permite acceder a valores mediante claves en tiempo constante

Conteo de frecuencias

Perfecto para contar ocurrencias de elementos en una colección

Live Coding

¿En qué consistirá la Demo?

Durante la demo, vamos a crear un sistema de registro de alumnos y sus calificaciones usando estructuras de datos. Aprenderemos a almacenar, modificar y consultar esta información utilizando listas y diccionarios.

1. *Crear una lista de diccionarios, donde cada diccionario representa a un alumno con:*
    a. *"nombre", "curso", "nota"*
2. *Agregar un nuevo alumno con append()*
3. *Modificar una nota accediendo a un diccionario por índice*
5. *Obtener estadísticas del grupo (por ejemplo, promedio)*
6. *Mostrar todos los nombres usando for y acceso por clave*
7. *Ordenar los alumnos por nota con sort() y una función lambda*

Tiempo: 30 Minutos

#Momentode Preguntas...

¿Qué diferencia hay entre una lista y un diccionario?

¿Cuándo usar un set en lugar de una lista?

¿Cómo accedo a un valor dentro de un diccionario?

¿Qué significa que una estructura sea “mutable”?

Momento:

Time-out!

⏰5 -10 min.

Ejercicio N° 1

Registro de alumnos con estructuras de datos

Registro de alumnos con estructuras de datos

Contexto: 🙌

Una institución educativa necesita registrar la información de sus estudiantes para poder visualizar, modificar y analizar sus calificaciones. Para eso, el equipo técnico decidió almacenar los datos usando estructuras de datos en Python que permitan una manipulación sencilla y eficiente.

Consigna: ✍️

Crear una rutina que utilice listas y diccionarios para almacenar, actualizar y analizar la información de un grupo de alumnos.

Tiempo 🕛**:** *35 Minutos*

Registro de alumnos con estructuras de datos

Paso a paso: ⚙️

1. Crear una lista llamada alumnos que contenga diccionarios, donde cada diccionario tenga tres claves:
2. "nombre", "curso", "nota"
3. Usar un bucle for para mostrar todos los nombres de los alumnos registrados.
4. Agregar un nuevo alumno al final de la lista utilizando el método append().
5. Modificar la nota del segundo alumno de la lista, accediendo por índice.
6. Calcular y mostrar el promedio general de las notas usando un bucle o una función auxiliar.
7. Ordenar la lista por nota de forma descendente utilizando el método sort() con una función lambda.

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Comprendimos cuándo y cómo usar listas y diccionarios***
- ✓ ***Aplicamos métodos básicos como append, pop, get, sort, etc.***
- ✓ ***Accedimos, modificamos y organizamos datos***
- ✓ ***Usamos estructuras combinadas (listas de diccionarios)***
- ✓ ***Creamos un sistema simple de gestión de datos en Python***

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son clave para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compártela o tráela al próximo encuentro sincrónico.

alkemy

¡Muchas gracias!

alkemy