Estructuras de dato en Python

Módulo: Fundamentos de programación Python para el análisis de datos

AE5: Aplicar las estructuras de dato de tipo colección del lenguaje python junto a sus características y utilidad para resolver un problema.

alkemy

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.

Este manual 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. Las listas permiten almacenar colecciones de elementos ordenados y modificables; los diccionarios ofrecen una estructura de clave-valor útil para búsquedas rápidas; las tuplas almacenan datos inmutables, ideales para colecciones constantes; y los sets eliminan elementos duplicados y permiten realizar operaciones de conjunto. Además, se explorará el concepto de compresión de listas, diccionarios y sets, una técnica avanzada que permite escribir código más compacto y optimizado.

Aprendizaje esperado

Cuando finalices la lección serás capaz de:

- Comprender qué es una estructura de datos y por qué son esenciales en la programación.
- Crear y manipular listas en Python, incluyendo el uso de listas anidadas y matrices.
- Crear y utilizar diccionarios en Python, incluyendo diccionarios anidados para datos complejos.
- Trabajar con tuplas, incluyendo técnicas de empaquetado y desempaquetado.
- Crear y realizar operaciones de conjunto con sets en Python.
- Aplicar técnicas de compresión para listas, diccionarios y sets para escribir código más eficiente y conciso.

Qué es una Estructura de Datos y Por Qué se Necesitan

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

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, lo cual es crucial en aplicaciones que manejan muchos datos o que requieren alta velocidad de procesamiento.

Clasificación de las Estructuras de Datos

En Python, las estructuras de datos se dividen en varios tipos según su naturaleza y propósito: secuencias (listas y tuplas), mapeos (diccionarios) y 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

Las estructuras de datos se utilizan en una amplia variedad de aplicaciones. Por ejemplo, una lista puede almacenar una secuencia ordenada de elementos, un diccionario es ideal para almacenar pares clave-valor, y un set es útil para eliminar duplicados y realizar operaciones de conjunto.

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

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.

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.

```
matriz = [[1, 2], [3, 4], [5, 6]]
print(matriz[1][0])  # Accede al elemento 3
```

Diccionarios

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. 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
```

Tuplas

Características de una Tupla

Las tuplas son estructuras de datos ordenadas e inmutables, lo que significa que no se pueden modificar después de su creación. Las tuplas son útiles para almacenar datos constantes que no deben cambiar, como coordenadas, y son más rápidas que las listas en términos de acceso. Esto las convierte en una opción eficiente cuando se necesita trabajar con conjuntos de datos fijos.

Creación de una Tupla

Para crear una tupla, se utilizan paréntesis `()` y los elementos se separan por comas. Las tuplas también pueden crearse sin paréntesis, usando solo comas.

```python
mi_tupla = (10, 20, "Python")
mi_tupla_sin_parentesis = 1, 2, 3
print(mi_tupla)  # Salida: (10, 20, 'Python')
print(mi_tupla_sin_parentesis)  # Salida: (1, 2, 3)
```

Acceso a Elementos en una Tupla

Se puede acceder a los elementos de una tupla mediante su índice, de la misma manera que en las listas.

```python
print(mi_tupla[1])  # Salida: 20
```

Empaquetado y Desempaquetado de Tuplas

El empaquetado permite agrupar múltiples valores en una tupla, mientras que el desempaquetado extrae estos valores en variables individuales.

```python
coordenadas = (4, 5)
x, y = coordenadas
print(x)  # Salida: 4
print(y)  # Salida: 5
```

Usos de las Tuplas en Programación

Las tuplas son especialmente útiles para representar datos inmutables, como fechas y puntos en un gráfico. También son comunes en funciones que devuelven múltiples valores en un solo retorno.

Sets

Características de un Set

Un set es una estructura de datos no ordenada que no permite elementos duplicados, lo cual lo convierte en una excelente opción para almacenar colecciones de datos únicos. Los sets son mutables, lo que significa que sus elementos se pueden añadir o eliminar, pero no permiten el acceso por índice debido a su naturaleza desordenada. Los sets son útiles para operaciones de conjunto, como la unión, intersección y diferencia.

Creación de un Set

Para crear un set, se utilizan llaves `{}` o la función `set()` para evitar crear un diccionario.

```
mi_set = {1, 2, 3, 4, 4, 5}
print(mi_set)  # Salida: {1, 2, 3, 4, 5} (elimina duplicados)
```

Agregar y Eliminar Elementos en un Set

Para agregar un elemento a un set, se utiliza el método `add()`. También se pueden eliminar elementos usando `remove()` o `discard()`.

```
mi_set.add(6)
mi_set.discard(2)
print(mi_set)  # Salida: {1, 3, 4, 5, 6}
```

Operaciones de Conjunto con Sets

Los sets permiten realizar operaciones de conjuntos como la unión (`|`), intersección (`&`), y diferencia (`-`), útiles para manejar datos únicos en diferentes colecciones.

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

print(set_a | set_b)  # Unión: {1, 2, 3, 4, 5}
print(set_a & set_b)  # Intersección: {3}
print(set_a - set_b)  # Diferencia: {1, 2}
```

Eliminación de Elementos Duplicados con Sets

Los sets se utilizan comúnmente para eliminar duplicados en una lista de datos.

```
lista = [1, 2, 2, 3, 4, 4, 5]
mi_set = set(lista)
print(list(mi_set))  # Salida: [1, 2, 3, 4, 5]
```

Compresión de Listas, Diccionarios y Sets

Compresión de Listas

La compresión de listas permite crear listas aplicando una operación o filtro a cada elemento de una colección existente en una sola línea de código, mejorando la eficiencia y la legibilidad.

```
numeros = [1, 2, 3, 4, 5]
cuadrados = [x**2 for x in numeros]
print(cuadrados)  # Salida: [1, 4, 9, 16, 25]
```

Compresión de Diccionarios

La compresión de diccionarios es similar a la de listas, permitiendo aplicar una operación o filtro a los pares clave-valor en una sola línea.

```
diccionario = {x: x**2 for x in range(5)}
print(diccionario)  # Salida: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Compresión de Sets

La compresión de sets elimina automáticamente duplicados y permite crear sets compactos y eficientes.

```
set_comp = {x for x in range(10) if x % 2 == 0}
print(set_comp)  # Salida: {0, 2, 4, 6, 8}
```

Cierre ✍️

Las estructuras de datos en Python son herramientas poderosas que permiten almacenar y manipular datos de manera eficiente. Al comprender las características y el uso adecuado de listas, diccionarios, tuplas y sets, los programadores pueden seleccionar la estructura más adecuada para cada situación, optimizando el rendimiento y organizando el código de manera más clara y modular. La compresión de listas, diccionarios y sets facilita la creación de colecciones derivadas en un solo paso, mejorando la eficiencia del código.

Con el dominio de estas estructuras de datos, los programadores pueden enfrentar desafíos complejos en el manejo de datos y desarrollar aplicaciones avanzadas y escalables. Practicar el uso de estas estructuras y técnicas de compresión es clave para mejorar en Python y para optimizar el procesamiento de datos en proyectos de cualquier escala.

Referencias 📚

Python Software Foundation. (s.f.). *The Python Standard Library*. https://docs.python.org/3/library/index.html

GeeksforGeeks. (s.f.). *Python Programming Language*. https://www.geeksforgeeks.org/python-programming-language

¡Muchas gracias!

Nos vemos en la próxima lección