UD07 - Apuntes – Estructuras de datos en Java

Contenido

Introducción:

Las estructuras de datos son una forma de organizar y almacenar datos para que puedan ser accedidos y trabajados de manera eficiente. Son fundamentales para la programación y son utilizadas en casi todos los programas o software.

Estructuras de datos en Python:

Las estructuras de datos principales son:

1.- **Listas:** Son colecciones ordenadas y mutables de elementos. Las listas son muy flexibles y pueden contener cualquier tipo de datos y pueden contener elementos duplicados.
2.- **Tuplas:** Son similares a las listas, pero son inmutables. Esto significa que una vez que una tupla es creada, no puede ser modificada.
3.- **Conjuntos (Sets):** Son colecciones no ordenadas de elementos únicos. Los conjuntos son ideales para cuando quieres evitar duplicados y no te importa el orden.
4.- **Diccionarios (Dicts):** Son colecciones no ordenadas de pares clave-valor. Son ideales para cuando necesitas asociar valores con claves para buscarlos de manera eficiente.

Estas estructuras de datos son importantes porque nos proporcionan formas de organizar y manipular nuestros datos de manera eficiente. Por ejemplo, si tienes una lista de elementos y necesitas buscar un elemento específico, es mucho más rápido buscarlo en un diccionario o en un conjunto (que tienen tiempo de búsqueda constante) que en una lista o en una tupla (que tienen tiempo de búsqueda lineal). Por lo tanto, elegir la estructura de datos correcta puede tener un impacto significativo en el rendimiento de tu programa.

Listas:

```
# Crear una lista
animales = ["Perro", "Gato", "Elefante"]

# Imprimir la lista
print("Lista:", animales)

# Añadir un elemento al final de la lista
animales.append("León")
print("Después de añadir un elemento:", animales)

# Insertar un elemento en una posición específica
animales.insert(1, "Tigre")
print("Después de insertar un elemento:", animales)

# Acceder a un elemento de la lista
print("El primer animal es:", animales[0])

# Modificar un elemento de la lista
animales[0] = "Lobo"
print("Después de modificar un elemento:", animales)

# Eliminar un elemento de la lista
animales.remove("Elefante")
print("Después de eliminar un elemento:", animales)

# Ordenar la lista
animales.sort()
print("Lista ordenada:", animales)

# Revertir el orden de la lista
animales.reverse()
print("Lista revertida:", animales)
```

Este código muestra cómo crear una lista, añadir elementos, insertar elementos en una posición específica, acceder a elementos, modificar elementos, eliminar elementos, ordenar la lista y revertir la lista. Las listas en Python son muy versátiles y son una de las estructuras de datos más utilizadas en Python.

Tuplas:

```
# Crear una tupla
animales = ("Perro", "Gato", "Elefante")

# Imprimir la tupla
print("Tupla:", animales)

# Acceder a un elemento de la tupla
print("El primer animal es:", animales[0])

# Intentar modificar un elemento de la tupla (esto dará un error)
try:
    animales[0] = "Lobo"
except TypeError:
    print("No puedes modificar una tupla")
```

```
# Crear una nueva tupla con un elemento añadido
animales = animales + ("León",)
print("Después de añadir un elemento:", animales)

# Crear una nueva tupla sin un elemento específico
animales = tuple(animal for animal in animales if animal !=
"Elefante")
print("Después de eliminar un elemento:", animales)
```

Este código muestra cómo crear una tupla, acceder a elementos, y que no puedes modificar una tupla una vez creada. Sin embargo, puedes crear una nueva tupla basada en una existente, añadiendo o eliminando elementos.

Conjuntos (Sets):

```
# Crear un conjunto
animales = {"Perro", "Gato", "Elefante"}

# Imprimir el conjunto
print("Conjunto:", animales)

# Añadir un elemento al conjunto
animales.add("León")
print("Después de añadir un elemento:", animales)

# Intentar añadir un elemento duplicado al conjunto (esto no hará
nada)
animales.add("León")
print("Después de intentar añadir un elemento duplicado:", animales)

# Verificar si un elemento está en el conjunto
contiene_leon = "León" in animales
print("¿Contiene 'León'?", contiene_leon)

# Eliminar un elemento del conjunto
animales.remove("Elefante")
print("Después de eliminar un elemento:", animales)

# Crear un conjunto a partir de una lista con elementos duplicados
lista = ["Perro", "Gato", "Perro", "Gato"]
conjunto = set(lista)
print("Conjunto a partir de lista con duplicados:", conjunto)
```

Este código muestra cómo crear un conjunto, añadir elementos, que los conjuntos no permiten elementos duplicados, verificar si un elemento está en el conjunto, eliminar elementos, y cómo crear un conjunto a partir de una lista con elementos duplicados (los duplicados se eliminan automáticamente).

Trabajando con Sets en clases personalizadas:

Si estás utilizando clases personalizadas en Python y necesitas comparar instancias de estas clases o usarlas en estructuras de datos como conjuntos (set) o como claves en diccionarios (dict), debes implementar algunos métodos especiales. Aquí están los más comunes:

- __eq__: Este método se utiliza para comparar dos objetos por igualdad. Debería devolver True si los objetos son iguales y False en caso contrario.
- __ne__: Este método se utiliza para comparar dos objetos por desigualdad. Normalmente, es suficiente definir __eq__ y Python usará este para inferir __ne__.
- __hash__: Este método se utiliza para obtener un valor hash del objeto. Este valor se utiliza en estructuras de datos como los conjuntos y los diccionarios para acelerar el acceso a los objetos. Si implementas __eq__, también deberías implementar __hash__, y asegurarte de que los objetos que son iguales según __eq__ tienen el mismo valor hash.

```
class Libro:
    def __init__(self, autor, titulo):
        self.autor = autor
        self.titulo = titulo

    def __eq__(self, otro):
        if isinstance(otro, Libro):
            return self.autor == otro.autor and self.titulo ==
otro.titulo
        return False

    def __hash__(self):
        return hash((self.autor, self.titulo))

    def __str__(self):
        return f"Libro(autor='{self.autor}', titulo='{self.titulo}')"

# Crear un conjunto de libros
libros = set()

# Añadir libros al conjunto
libros.add(Libro("Autor1", "Titulo1"))
libros.add(Libro("Autor2", "Titulo2"))
libros.add(Libro("Autor3", "Titulo3"))

# Imprimir el conjunto
for libro in libros:
    print(libro)

# Crear un diccionario con libros como claves
libros_dict = {
    Libro("Autor1", "Titulo1"): "Libro1",
    Libro("Autor2", "Titulo2"): "Libro2",
    Libro("Autor3", "Titulo3"): "Libro3",
}

# Imprimir el diccionario
for libro, valor in libros_dict.items():
    print(f"{libro} => {valor}")
```

En este código, la clase Libro implementa __eq__ para comparar libros por autor y título, y __hash__ para obtener un valor hash de los libros. Con estos métodos, puedes usar instancias de Libro en conjuntos y como claves en diccionarios. También implementa __str__ para una representación legible de los libros.

Diccionarios (Dicts):

```
# Crear un diccionario
animales = {"Perro": "Mamífero", "Gato": "Mamífero", "Elefante":
"Mamífero"}

# Imprimir el diccionario
print("Diccionario:", animales)

# Añadir un elemento al diccionario
animales["León"] = "Mamífero"
print("Después de añadir un elemento:", animales)

# Acceder a un elemento del diccionario
print("El Perro es un:", animales["Perro"])

# Verificar si un elemento está en el diccionario
contiene_leon = "León" in animales
print("¿Contiene 'León'?", contiene_leon)

# Eliminar un elemento del diccionario
del animales["Elefante"]
print("Después de eliminar un elemento:", animales)

# Obtener todas las claves del diccionario
claves = animales.keys()
print("Claves:", claves)

# Obtener todos los valores del diccionario
valores = animales.values()
print("Valores:", valores)

# Obtener todos los pares clave-valor del diccionario
items = animales.items()
print("Items:", items)
```

Este código muestra cómo crear un diccionario, añadir elementos, acceder a elementos, verificar si un elemento está en el diccionario, eliminar elementos, y cómo obtener las claves, los valores y los pares clave-valor del diccionario. Los diccionarios en Python son muy útiles y son una de las estructuras de datos más utilizadas en Python.

Trabajando con Dicts en clases personalizadas:

```
class Libro:
    def __init__(self, autor, titulo):
        self.autor = autor
        self.titulo = titulo

    def __eq__(self, otro):
        if isinstance(otro, Libro):
            return self.autor == otro.autor and self.titulo ==
otro.titulo
```

```
        return False

    def __hash__(self):
        return hash((self.autor, self.titulo))

    def __str__(self):
        return f"Libro(autor='{self.autor}', titulo='{self.titulo}')"

# Crear un diccionario con libros como claves
libros_dict = {
    Libro("Autor1", "Titulo1"): "Resumen del libro 1",
    Libro("Autor2", "Titulo2"): "Resumen del libro 2",
    Libro("Autor3", "Titulo3"): "Resumen del libro 3",
}

# Imprimir el diccionario
for libro, resumen in libros_dict.items():
    print(f"{libro} => {resumen}")

# Acceder a un elemento del diccionario
libro1 = Libro("Autor1", "Titulo1")
print(f"Resumen del libro 1: {libros_dict[libro1]}")

# Verificar si un libro está en el diccionario
contiene_libro1 = libro1 in libros_dict
print(f"¿Contiene el libro 1? {contiene_libro1}")
```

En este código, la clase Libro implementa __eq__ y __hash__, lo que permite usar instancias de Libro como claves en un diccionario. El diccionario libros_dict mapea libros a sus resúmenes. El código muestra cómo crear un diccionario con libros como claves, imprimir el diccionario, acceder a elementos y verificar si un libro está en el diccionario.