Estructuras de datos en Python

Las estructuras de datos son formas de organizar y almacenar datos en un programa de manera que puedan ser utilizados eficientemente. En Python, las principales estructuras de datos nativas incluyen

- listas
- tuplas
- conjuntos
- diccionarios

Cada una con características específicas que las hacen útiles para diferentes tareas.

Array

un array es una estructura de datos que almacena elementos de un mismo tipo en una secuencia contigua de memoria.

```
import array
mi_array = array.array('i', [1, 2, 3, 4, 5])  # 'i' indica que son enteros
print(mi_array[2])  # 3
mi_array.append(6)  # Agregar un elemento
```

Listas

Las listas son colecciones **ordenadas y mutables** que permiten almacenar elementos de cualquier tipo de dato. Son una de las estructuras más flexibles y ampliamente usadas en Python.

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

# Adición de elementos:
mi_lista.append(6)  # Agrega un elemento al final
mi_lista.insert(2, 10)  # Inserta el número 10 en la posición 2

# Eliminación de elementos:
mi_lista.remove(3)  # Elimina el primer valor igual a 3
elemento = mi_lista.pop(1)  # Elimina y devuelve el elemento en la posición 1
del mi_lista[0]  # Elimina el primer elemento de la lista

# Ordenación
mi_lista.sort()
```

Ejemplo: crear lista de cuadrados

```
squares = []
for x in range(10):
```

```
    squares.append(x**2)
```

Usando listas como pilas

Las listas pueden funcionar como **pilas** (estructura **LIFO**: Last In, First Out). El último elemento que se añade es el primero que se retira.

```
pila = []
pila.append('a')
pila.append('b')
elemento = pila.pop()  # Devuelve y elimina 'b'
```

Usando listas como colas

Para usar listas como colas (**FIFO**: First In, First Out), donde el primer elemento que entra es el primero que sale, es más eficiente utilizar la colección deque de la librería collections.

```
from collections import deque
cola = deque([1, 2, 3])
cola.append(4)  # Agrega 4 al final de la cola
primero = cola.popleft()  # Elimina y devuelve el primer elemento (1)
```

La instrucción del

La instrucción del se utiliza para eliminar elementos de una estructura de datos o borrar variables.

```
lista = [1, 2, 3, 4, 5]
del lista[2]  # Elimina el tercer elemento
del lista  # Elimina la variable lista
```

Tuplas y secuencias

Las tuplas son colecciones **ordenadas** pero **inmutables**, lo que significa que no se pueden modificar después de su creación. Son útiles cuando se necesita una secuencia constante de elementos.

```
mi_tupla = (1, 2, 3, 4)
Acceso a elementos:
python
Copiar código
valor = mi_tupla[1]  # Devuelve 2
```

Como las tuplas son inmutables, no es posible agregar ni eliminar elementos directamente. Si se necesita modificar una tupla, se debe crear una nueva tupla a partir de la existente.

Conjuntos

Los conjuntos son colecciones **no ordenadas** de elementos **únicos**. Permiten realizar operaciones matemáticas como la unión, intersección y diferencia.

```
mi_conjunto = {1, 2, 3}
# Adición de elementos:
mi_conjunto.add(4)  # Agrega el valor 4

# Eliminación de elementos:
mi_conjunto.remove(2)  # Elimina el valor 2

# Unión, intersección y diferencia:
conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}
union = conjunto1 | conjunto2  # {1, 2, 3, 4, 5}
interseccion = conjunto1 & conjunto2  # {3}
diferencia = conjunto1 - conjunto2  # {1, 2}
```

Diccionarios

Los diccionarios son colecciones de pares clave-valor. Son útiles para mapear una clave única a un valor.

```
#Creación
mi_diccionario = {'nombre': 'Juan', 'edad': 30}

# Adición y modificación de elementos:

mi_diccionario['ciudad'] = 'Madrid'  # Agrega una nueva clave-valor
mi_diccionario['edad'] = 31  # Modifica el valor asociado a la clave 'edad'

# Eliminación de elementos:
del mi_diccionario['ciudad']  # Elimina la clave 'ciudad'
valor = mi_diccionario.pop('edad')  # Elimina y devuelve el valor as
```