SESIÓN ESTRUCTURAS DE DATOS EN PYTHON

CONTENIDOS:

- ¿Qué es una estructura de datos y por qué se necesitan?
- Listas: características, creación, agregar elementos, rescate de elementos, rescate de un rango de elementos, listas y cadenas de caracteres, listas anidadas y matrices.
- Diccionarios: características, creación, agregar elementos, rescate de elementos, diccionarios anidados.
- Tuplas: características, creación, rescate de elementos.
- Empaquetado y desempaquetado de tuplas.
- Sets: características, creación, operaciones de conjunto.
- Compresión de listas, diccionarios y sets.

¿QUÉ ES UNA ESTRUCTURA DE DATOS Y POR QUÉ SE NECESITAN?

Las estructuras de datos son formas de organizar y almacenar información en un programa, permitiendo que sea utilizada de manera eficiente. Facilitan el acceso y la manipulación de los datos y son esenciales para optimizar el rendimiento, reducir la complejidad y hacer el código más claro y mantenible.

LISTAS

Una lista en Python es una estructura de datos que almacena una secuencia ordenada de elementos, que pueden ser de distintos tipos (números, cadenas, etc.). Las listas son mutables, lo que significa que se pueden modificar agregando, eliminando o cambiando elementos. Los elementos se acceden mediante índices, comenzando desde 0, y permiten organizar y manipular datos de forma flexible en un programa.

¿Qué se puede hacer con las listas?

- **Crear listas:** usando una sintaxis simple con el nombre de la lista y sus datos “nombre_lista = [dato1, dato2, dato3]”.

- **Agregar elementos:** usando el método append() puedes agregar elementos al final de la lista.
- **Rescate de elementos:** acceder a elementos utilizando su índice. Las listas comienzan por el índice 0.
- **Rescate de un rango de elementos:** se utiliza la notación [inicio:fin] para obtener una sublista de la lista original.
- **Lista y cadena de caracteres:** las listas pueden almacenar cadenas, y se pueden dividir cadenas en listas usando el método splits().
- **Listas anidadas y matrices:** las listas pueden contener otras listas, útiles para representar matrices.

```
mi_lista = [1, 2, 3, 4]  # Creación de una lista

mi_lista.append(5) # Uso de método append() para agregar un elemento al final de la lista

elemento = mi_lista[0]  # Acceder al elemento por el indice, para este ejemplo al Primer elemento

sublista = mi_lista[1:3] #La anotación [inicio:fin] permite una sublista de un rango de elementos

cadena = "Hola Mundo" # cadena se almacena en lista_cadena y se divide usando el método split()
lista_cadena = cadena.split()  # ['Hola', 'Mundo']


matriz = [[1, 2], [3, 4]] #Lista puede contener otras listas, representando matrices
```

Ilustración 1 Ejemplos con listas

DICCIONARIOS

Un diccionario en Python es una estructura de datos que almacena pares de valores en formato clave-valor. Cada elemento tiene una clave única que actúa como identificador y un valor asociado. Esta estructura permite acceder rápidamente a los datos mediante sus claves en lugar de utilizar índices, como en las listas. Los diccionarios son mutables, por lo que se pueden agregar, modificar o eliminar pares clave-valor en cualquier momento. Se crean utilizando llaves {}, y las claves suelen ser cadenas, números u otros tipos inmutables. Los diccionarios son útiles para representar relaciones entre elementos, como en datos tabulares o cuando se necesita acceso rápido a valores específicos.

¿Qué se puede hacer con los diccionarios?

- **Crear Diccionarios:** usando una sintaxis simple con el nombre del diccionario y la clave y su valor "nombre_diccionario = {clave: valor, clave: valor}
- **Agregar elementos:** se añade un nuevo par clave-valor indicando la clave y el valor almacenado.
- **Rescatar elementos:** acceder al valor de un elemento a través de su clave.
- **Diccionarios anidados:** los diccionarios pueden contener otros diccionarios para organizar datos más complejos.

```
mi_diccionario = {'nombre': 'Ana', 'edad': 25} # Creación Diccionario

mi_diccionario['ciudad'] = 'Santiago' # Agregar Elementos

edad = mi_diccionario['edad'] # Rescate de Elementos

usuarios = {                          # Diccionarios Anidados
    'usuario1': {'nombre': 'Ana', 'edad': 25},
    'usuario2': {'nombre': 'Luis', 'edad': 30}
}
```

Ilustración 2 Ejemplos diccionarios

TUPLAS

Una tupla en Python es una estructura de datos que permite almacenar una secuencia ordenada de elementos, similares a una lista, pero es inmutable, lo que significa que no se pueden modificar, agregar o eliminar sus elementos después de creada. Las tuplas se definen con paréntesis () y pueden contener distintos tipos de datos (números, cadenas, etc.). Su inmutabilidad las hace más eficientes y seguras para almacenar datos constantes. Las tuplas son útiles cuando se desea asegurar que una colección de valores permanezca sin cambios a lo largo del programa.

¿Qué se puede hacer con las tuplas?

- **Creación de tuplas:** Con una sintaxis simple indicando el nombre de la tupla y sus valores almacenados "nombre_tupla = (valor1, valor2, valor3).
- **Rescate de elementos:** al igual que con las listas, se accede a los elementos por sus índices.

```
mi_tupla = (1, 2, 3) # Creación Tupla

primer_elemento = mi_tupla[0]  #Rescate Elementos
```

Ilustración 3 Ejemplo tuplas

EMPAQUETADO Y DESEMPAQUETADO DE TUPLAS

El empaquetado y desempaquetado de tuplas en Python son técnicas útiles para trabajar con múltiples valores de manera compacta y directa.

Empaquetado de tuplas ocurre cuando se asignan varios valores a una sola variable en forma de tupla, sin necesidad de usar paréntesis. Python agrupa automáticamente los valores en una tupla, lo que simplifica la asignación. Por ejemplo:

```
tupla = 1, 2, 3
```

Ilustración 4 Empaquetado tupla

Aquí, tupla contiene los valores (1, 2, 3).

Desempaquetado de tuplas permite asignar cada valor de una tupla a una variable individual, facilitando el acceso directo a cada elemento sin índices. Esto se hace igualando la tupla a varias variables en el mismo orden:

```
a, b, c = tupla
```

Ilustración 5 Desempaquetado tupla

En este caso, a obtendrá 1, b obtendrá 2, y c obtendrá 3.

Esta técnica es útil cuando se quieren asignar múltiples valores de manera rápida, especialmente en funciones que devuelven varios resultados. También ayuda a hacer el código más claro y organizado.

SETS

Un set en Python es una colección desordenada de elementos únicos, lo que significa que no permite duplicados. Los sets se crean usando llaves {} o la función set(), y se utilizan para almacenar elementos sin un orden específico. Debido a su estructura, los sets son eficientes para realizar operaciones como pruebas de pertenencia, unión, intersección y diferencia entre conjuntos. A diferencia de las listas o tuplas, los elementos de un set deben ser inmutables (como números o cadenas), pero el set en sí es mutable, permitiendo agregar o eliminar elementos. Son ideales para operaciones de agrupamiento o comparación sin duplicados.

Para crear un set, se usa una sintaxis simple indicando el nombre y sus valores "nombre_set = {valor1, valor2, valor3}.

```
mi_set = {1, 2, 3}  # Creación de un set
```

Ilustración 6 Creación set

Los sets permiten realizar operaciones de conjunto, es decir, operaciones matemáticas como unión, intersección y diferencia.

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

# Unión
union = set_a | set_b  # {1, 2, 3, 4, 5}

# Intersección
interseccion = set_a & set_b  # {3}

# Diferencia
diferencia = set_a - set_b  # {1, 2}
```

Ilustración 7 Operaciones Set

COMPRESIÓN DE LISTAS, DICCIONARIOS Y SETS

La compresión de listas, diccionarios y sets en Python es una técnica para crear estas estructuras de datos de manera concisa y eficiente. Mediante la compresión, se generan listas, diccionarios o sets en una sola línea usando una expresión compacta, que suele incluir un bucle y condiciones opcionales. Por ejemplo, una lista de cuadrados puede crearse como [x**2 for x in range(10)]. La compresión mejora la legibilidad y el rendimiento del código, especialmente al trabajar con datos filtrados o transformados, facilitando el manejo de grandes colecciones de elementos.

Compresión de listas: es una forma concisa de crear listas aplicando operaciones a cada elemento.

```python
lista = [x * 2 for x in range(5)]  # [0, 2, 4, 6, 8]
```

Ilustración 8 Compresión lista

Compresión de diccionarios: crear diccionarios rápidamente usando una expresión y clave-valor.

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

Ilustración 9 Compresión diccionario

Compresión de sets: similar a las listas, pero en sets.

```python
set_comprimido = {x * 2 for x in range(5)}  # {0, 2, 4, 6, 8}
```

Ilustración 10 Compresión set