Check Point 4. Parte Teórica

1. **¿Cuál es la diferencia entre una lista y una tupla en Python?**

Una tupla es una sucesión de elementos de cualquier tipo (o varios tipos) separados por comas. En cuanto a su sintaxis, se definen usando paréntesis **()**. No se pueden añadir ni eliminar elementos de una tupla una vez creada; son objetos inmutables, similares a las cadenas o los números.

```
tupla = ("primero", "segundo", "tercero")
```

Las listas son secuencias ordenadas de elementos, al igual que las tuplas, pero con la diferencia de que pueden modificarse; son objetos mutables. Esto significa, que se les puede añadir, modificar o eliminar elementos. Las listas se declaran indicando los elementos en orden, separados por comas, y encerrados entre corchetes **([])**.

```
lista = ["primero", "segundo", "tercero"]
```

Se utiliza una lista cuando se necesita una colección de elementos que se modificarán con el tiempo, por ejemplo, una lista de productos en una tienda de la que se irán añadiendo y quitando artículos.

Por otro lado, se utiliza una tupla cuando se desea que los elementos sean inmutables, como en el caso de coordenadas o listas de códigos postales.

Una diferencia destacable entre ambas colecciones está en la memoria que ocupan. Las tuplas son más eficientes debido a que son inmutables y, por lo tanto, necesitan menos espacio de memoria.

Las listas tienen métodos incorporados que permiten agregar, eliminar, y modificar elementos, como 'append()', 'remove()', 'insert()', 'copy()',' index()',' sort()', entre otros.

Las tuplas tienen menos métodos disponibles debido a su inmutabilidad, pero aun así pueden utilizar funciones como count() e index().

2. **¿Cuál es el orden de las operaciones?**

El orden en que Python realiza las operaciones es el mismo que en las reglas aritméticas:

1. **Paréntesis**, que es la prioridad más alta. Los paréntesis pueden ser anidados para indicar el orden de evaluación deseado. Por ejemplo,

en la expresión **resultado = (5 + 3) * 2,** primero se suman 5 y 3, y luego el resultado se multiplica por 2, dando como resultado 16.

2. Exponentes. Por ejemplo, en la expresión **resultado = 2 ** 3**, se eleva 2 a la potencia de 3, dando como resultado 8.

3. Multiplicación y División, en el orden en el que aparezcan, de izquierda a derecha. Por ejemplo, en la expresión **resultado = 10 / 2 * 3**, primero se divide 10 por 2, y luego el resultado se multiplica por 3, dando como resultado 15.

4. Suma y Resta, Al igual que la multiplicación y la división, estas operaciones se evalúan de izquierda a derecha según aparecen en la expresión.

Ejemplo:

2+3*4 → 2+(3*4) →2+12 →14

3. ¿Qué es un diccionario Python?

Un diccionario para Python es una estructura de datos que permite almacenar pares clave-valor. Cada clave en un diccionario debe ser única y se utiliza para acceder al valor asociado. Al definir un diccionario, los pares clave-valor se separan con dos puntos (:) y los pares se separan por comas (,). El conjunto se delimita por llave ({}). Por ejemplo,

```
diccionario = { "clave1": "valor", "clave2": "otro_valor"}
cliente= {"nombre": "Ana", "edad": "52"}
```

Los diccionarios son similares a las listas en el sentido de que son objetos mutables, lo que significa que se pueden modificar después de su creación, puedes agregar nuevos pares clave-valor, eliminar pares existentes o actualizar los valores de las claves existentes. Ejemplo:

```
cliente["correo"]=ana.m.pinon@gmail.com #agrega un nuevo par clave-valor
cliente["edad"]=53 # actualiza el valor de la clave existente
```

El uso de diccionarios es común en Python para representar asociaciones entre diferentes tipos de datos, como nombres y edades, palabras y definiciones, o

identificadores y objetos. Proporcionan un acceso rápido a los valores a través de las claves y son muy eficientes para realizar búsquedas y recuperar datos en grandes conjuntos de información.

4. **¿Cuál es la diferencia entre el método ordenado y la función de ordenación?**

El método **'sort()'** y la función **'sorted()'** en Python son dos formas de ordenar elementos en una lista, pero difieren en cómo afectan a la lista original y en cómo devuelven los resultados.

El método **'sort()'** se aplica directamente a la lista y reorganiza sus elementos en su lugar, es decir, modifica la lista original. Dependiendo de que tipo de valores sean estos, puede hacerlo alfabéticamente ('strings') o por su valor ('integers').

Funciona únicamente con listas y no devuelve ningún valor explicito, ya que simplemente ordena la lista sobre la que se llama.

La sintaxis básica es '**lista.sort()**' y puede tomar un parámetro opcional **'reverse'** para ordenar en sentido inverso.

Ejemplo:

```
numeros = [3,1,4,5,9]
numeros.sort()
print(numeros) → # [1,3,4,5,9]
```

La función '**sorted()'** toma cualquier iterable como argumento y devuelve una nueva lista que contiene los elementos, sin modificar la lista original.

No modifica la lista original, sino que genera una nueva lista ordenada.

La sintaxis básica es **'sorted(iterable)',** donde 'iterable' puede ser una lista, tupla, diccionario, etc.

Puede tomar los mismos parámetros opcionales que '**sort()'**, como 'reverse'.

Ejemplo:

```
numeros = [3,1,4,5,9]
numeros_ordenados=sorted.numeros
print(números_ordenados) → # [1,3,4,5,9]
```

Ambos métodos son útiles dependiendo de si se desea modificar la lista original o mantenerla intacta. Si se necesita la lista original ordenada, se puede usar 'sort()',

mientras que si se quiere conservar la lista original y obtener una nueva lista ordenada, se debe utilizar 'sorted ()'.

5. **¿Qué es un operador de reasignación?**

Un operador de reasignación es una construcción en lenguajes de programación que combina un operador aritmético con un operador de asignación para realizar un cálculo mientras se realiza una asignación de valor a una variable. En esencia, simplifica la tarea de modificar una variable con una operación aritmética y luego asignar el resultado a la misma variable.

Hay diferentes tipos de operadores de asignación, los más utilizados en Python son:

- **+=** → suma y asignación, agrega el valor a la variable y luego asigna el resultado a la misma variable.
- **-=** → resta y asignación, resta el valor de la variable y luego asigna el resultado a la misma variable.
- ***=** → multiplicación y asignación.
- **/=** → división y asignación.
- **%=** → modulo y asignación, realiza la operación de módulo entre el valor de la variable y luego asigna el resultado a la misma variable.
- **//=** → división entera y asignación.
- ****=** → potencia y asignación

Estos operadores son útiles para abreviar y hacer más legible el código cuando se necesita realizar una operación aritmética y asignar el resultado a la misma variable.

Ejemplo:

```
numero=5
numero +=3 # Equivalente a numero= numero + 3
print(numero) → # 8
```