Checkpoint 4

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

La principal diferencia entre una tupla y una lista, es que las listas, son conjuntos de elementos mutables, es decir, que pueden ser cambiados, alterados o eliminados después de haber sido creada, por el contrario la tupla es un conjunto de elementos inmutables, por lo que una vez creado no se puede modificar, ni añadir elementos. La única forma de añadir elementos a un tupla sería usando la reasignación, pero este paso nos crea un nueva tupla, no modifica la original.

Además de la mutabilidad, existen otras diferencias como por ejemplo la sintaxis, para hacer una lista utilizamos “[ ]” mientras que para hacer una tupla la sintaxis utilizada es “( )”. Un ejemplo de lista y tupla sería:

lista = [“Rubens”, “Ballester”, “Lillo]
tupla = (“Rubens”, “Ballester”, “Lillo”)

En cuanto a términos de rendimiento, gracias a su inmutabilidad las tuplas suelen ser más eficientes que las listas sobre todo en acciones como la iteración y el acceso a los elementos.

En general se recomienda el uso de listas cuando queramos tener un conjunto de elementos que queremos ir modificando a posteriori y el uso de las tuplas para cuando queramos tener un conjunto de elementos inmutable y que no pueda sufrir cambios.

2. ¿Cuál es el orden de las operaciones?

En Python, el orden de las operaciones sigue las reglas estándar de los operadores matemáticos. Estas reglas dictan el orden en que se llevan a cabo las operaciones.

El orden de las operaciones en Python es el siguiente:

1. Paréntesis.
2. Potencias.
3. Multiplicación.
4. División.
5. Suma.
6. Resta.

Por ejemplo en el siguiente código, el orden que seguiría el programa es:

resultado = 2 + 3 * 4 # En primer lugar realiza la multiplicación “3 * 4” y al resultado de esta (12) le suma el 2 “2 + 12”, dando como resultado 14.

Si en el código anterior quisiéramos que la suma sea realizada antes que la multiplicación, al igual que en las matemáticas convencionales, podemos usar los paréntesis para dar prioridad a la operación ya que como hemos visto anteriormente el paréntesis es la primera operación en ser realizada. Entonces:

resultado = (2 + 3) * 4 # En primer lugar realiza la suma “2 + 3” y al resultado de esta (5) le multiplica el 4, dando como resultado 20.

Como podemos observar por la diferencia de resultados, es importante tener en cuenta estas reglas al escribir expresiones en Python para garantizar que la lógica y el resultado sean los que queremos.

3. ¿Qué es un diccionario Python?

Un diccionario es un elemento de python que nos permite almacenar por pares, una clave y su valor para posteriormente poder acceder a ese valor con esa clave única. Los diccionarios son mutables y podemos añadir, eliminar o modificar elementos después de ser creado.
Las claves en un diccionario deben ser únicas, pero los valores pueden ser cualquier tipo de datos (números, cadenas, listas, tuplas, otros diccionarios, etc.). Los diccionarios permiten un acceso rápido a los valores a través de sus claves. Esto significa que puedes recuperar el valor asociado con una clave de forma rápida, independientemente del tamaño del diccionario.

La sintaxis utilizada para crear un diccionario es “{ }” y un ejemplo de diccionario sería:

```
diccionario = {
 "nombre": "Rubens",
 "edad": 31,
 "ciudad": "Bilbao"
      }
```

En este ejemplo, "nombre", "edad" y "ciudad" son las claves, y "Rubens", 31 y "Bilbao" son los valores que les hemos asignado. Los diccionarios son muy útiles en Python para distintas tareas como mapear claves a valores, representar datos estructurados y realizar búsquedas eficientes.

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

En Python, existen dos maneras de ordenar una secuencia de elementos, como una lista o una tupla: mediante el método sort() y con la función sorted(). Aunque ambos realizan la misma tarea básica de ordenar los elementos, lo hacen de forma distinta tanto en la forma de operar como en la forma en que afectan a los datos originales.

1. El método sort() se aplica directamente a la lista y modifica la original ordenando los elementos. Este método solo puede ser usado en listas. Un ejemplo sería:

```
lista = [3, 1, 4, 1, 5, 9,]
lista.sort()
print(lista)  # Salida: [1, 1, 3, 4, 5, 9]
```

2. La función sorted() puede ser utilizada tanto en listas, tuplas, colecciones, etc. Esta función toma como argumento la lista que queramos ordenar y nos devuelve una nueva lista ordenada pero sin haber modificado la original, por ejemplo:

```
tupla = (3, 1, 4, 1, 5, 9)
lista_ordenada = sorted(tupla)
print(lista_ordenada)  # Salida: [1, 1, 3, 4, 5, 9]
```

En resumen, la principal diferencia está en cómo afectan a los datos originales, el método sort() modifica la lista original, mientras que sorted() devuelve una nueva lista ordenada dejando los datos originales intactos. Por lo tanto elegiremos una u otra en función del tipo de dato que estemos manejando y de si queremos modificar los datos originales o no.

5. ¿Qué es un operador de asignación?

Los operadores de asignación son elementos que usa Python para asignar un valor a una variable, el operador más utilizado es el “=”, por ejemplo, “ num = 5”, en esta ocasión el operador “=” está asignando el valor 5 a la variable num. En Python existen otros operadores que nos ayudan a asignar o reasignar valores a las variables, entre los operadores más comunes podemos encontrar:

1. “+=” Suma.
2. “-=” Resta.
3. “*=” Multiplicación.
4. “/=” División.
5. “//=” División Entera.
6. “**=” Exponente.
7. “%=” Módulo.

Además podemos usar los operadores de asignación para sumar dos variables ya definidas, como por ejemplo:

```
total = 100
producto = 120
total += producto   # Esto equivale a total = total + producto
print(total)   # Salida: 220
```

En este ejemplo, mediante el operador “=” asignamos el valor 100 a la variable total y el valor 120 a la variable producto. Lo que nos permiten estos operadores es que mediante este código “total += producto” podemos sumar el valor de la variable producto, a la variable total y almacenar el resultado en esa variable.