Checkpoint 4

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

Ambos tipos de datos pueden almacenar varios elementos en una única variable. Las listas se crean rodeando los elementos con corchetes [] mientras que las listas se crean usando paréntesis ().

```
my_list = ['My', 'First', 'List']

my_tuple = ('This', 'is', 'a tuple')
```

La mayor diferencia entre lisa y tupla es que mientras que las tuplas son *inmutables,* las listas son *mutables*. Es decir, cuando se crea una tupla, sus elementos no pueden cambiar, la tupla no puede ser alterada, así, si se intenta cambiar el valor de sus elementos o se añade uno nuevo, se obtendrá un error.

Por otro lado, las listas son *mutables*, el valor de sus elementos se puede cambiar o eliminar y se pueden añadir nuevos elementos.

Se recomienda usar tuplas cuando se quiere que la variable sea solo de lectura, cuando no tiene que ser cambiada. Las listas, en cambio, son útiles cuando se quieren datos flexibles, modificables cuando sea necesario.

2. ¿Cuál es el orden de las operaciones?

El orden de operaciones se puede seguir usando PEMDSR

‘P’: Paréntesis ()
‘E’: Exponente **
‘M’: Multiplicación *
‘D’: División /
‘S’: Suma +
‘R’: Resta –

Ejemplo:

8 + 2 * 5 - (9 + 2) ** 2
8 + 2 * 5 - 11 ** 2
8 + 2 * 5 - 121
8 + 10 - 121
-103

3. ¿Qué es un diccionario Python?

Los diccionarios se usan para almacenar elementos en parejas de (clave:valor). Así, los valores son accesibles mediante la clave del elemento. Son muy útiles para organizar los datos en forma de pares.

El diccionario es mutable, es decir, se pueden añadir o eliminar elementos o cambiarlos. Sin embargo, el diccionario no permite claves duplicadas, por ejemplo, dado el diccionario de la imagen, se ve que 'Jone' está duplicado, por lo tanto, si se imprime el diccionario, se ve que el ultimo item ('Jone': 25) no está incluido.

```
my_dictionary = {
    'Andy' : 26,
    'Jone': 28,
    'Aritz': 20,
    'Jone': 25
}
```

Print:

```
{'Andy': 26, 'Jone': 25, 'Aritz': 20}
```

No obstante, el valor (en este caso la edad) puede ser igual para diferentes claves. En el siguiente ejemplo se puede ver que 'Andy' y 'Leire' tienen el mismo valor (26).

```
my_dictionary = {
    'Andy' : 26,
    'Jone': 28,
    'Aritz': 20,
    'Leire': 26
}
```

Print:

```
{'Andy': 26, 'Jone': 28, 'Aritz': 20, 'Leire': 26}
```

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

Método ordenado .sort(): modifica la lista original de manera que los elementos son ordenados alfabéticamente o en caso de números de menos a mayor. Este método es únicamente válido para listas.

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

Función de ordenación .sorted(): no modifica la lista original, devuelve una lista ordenada, por lo tanto, se requiere una nueva variable para guardar la lista ordenada. Esta función puede ser empleada en otros tipos de datos como tuplas o diccionarios.

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

5. ¿Qué es un operador de reasignación?

Son operadores que se utilizan para modificar el valor de una variable, se le asigna un nuevo valor teniendo en cuenta su valor anterior. En el siguiente ejemplo se puede ver que teniendo x un valor inicial de 10, se le suma 3, lo que da como resultado x = 13

```
x = 10
x += 3
print(x) # 13
```

En la siguiente tabla se pueden ver distintos operadores de asignación (https://www.w3schools.com/python/python_operators.asp)

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)