QUESTIONS CHECKPOINT 4

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

Las tuplas y las listas son dos de los 4 tipos de datos usados en Python, que pueden almacenar una colección de uno o más elementos.

tuplas:

Son un un tipo de datos que permite guardar información, la gran ventaja de las tuplas
Una de las diferencias es la sintaxis:
Las tuplas se crean mediante paréntesis ()
mi_tupla = ()
mi_tupla = (5, 55, 9, ¨Hi¨, True)
Print(tupla[1] # 55
Utilizo tuplas cuando necesito una colección de elementos **ordenados e inmutables**.
Las tuplas con inmutables no se pueden cambiar ni modificar, pueden contener elementos de diferentes tipos de datos
Con las tuplas se pueden usar como claves de diccionarios.(son hashables)

Para ecceder a los elementos de una tupla se utiliza un indice numérico que empieza por 0.

```
mi_tupla = (100, 200, 300, 450, 520)
print(mi_tupla[0])  # Resultado: 100
print(mi_tupla[2])  # Resultado: 300
```

Podemos asignar elementos de una tupla a variables individuales esto es conocido como **desempacamiento de tuplas**.

```
mi_tupla = ("Luis", "Rodriguez", 60)
mi_tupla = nombre, apellido, edad
print(nombre)  # Resultado: Luis
print(apellido)  # Resultado: Rodriguez
print(edad)  # Resultado: 60
```

listas:

- Las listas se crean usando corchetes [] separando a sus elementos con una coma, la ventaja es que se puede almacenar tipos de datos diferentes.

```
mi_list = []
mi_list = [10, 20, 20, ¨Python¨, False]
```

También se puede crear usando **list** y pasando un objeto iterable.

lista = list(«10, 20, 20»)

Las listas son mutables y dinámicas pueden agregar, quitar elementos y realizar cualquiera de ellas sobre la marcha.
Las listas son ordenadas es decir mantienen el orden que han sido definidas, se pueden anidar es decir meter una dentro de la otra.

Se puede acceder y modificar las listas.

ejemplo: tenemos una lista **b** con 3 elementos almacenados y podemos **acceder** a ellas de la siguiente manera:

```
b = [20, «BootCamp», 3.1416]
print(b[0])   #20
print(b[1])   #BootCamp
print(b[2])   #3.1416
```

Ejemplo: si queremos **modificar** un elemento de la lista solo se asigna con el operador = el nuevo valor

```
b[2] = 8
print(b) #[20, «BootCamp», 8]
```

2. ¿ Cuál es el orden de las operaciones?

La jerarquía de los operadores determina el orden que se resuelven las expresiones de las operaciones matemáticas.
Es muy simple para poder recordar usando **PEMDAS**
P significa paréntesis,
E significa exponencial,
MD significa multiplicación y división y
AS significa suma y resta

3. ¿Qué es un diccionario en Python?

Un diccionario son una estructura de datos en Python te permite almacenar pares: **key:value**
key son únicas e inmutables, como una cadena de texto o un numero.
value pueden ser modificados después de su creación, el valor puede ser cualquier objeto válido en Python.
Los diccionarios se definen por corchetes {} y los pares clave-valor y se separan por dos puntos **:**

Ejemplo:
mi_diccionario = {¨nombre¨: ¨Sara¨, ¨altura¨: 160, ¨ documen-to¨:10050101}

key = nombre
valor = sara

Los diccionarios en Python permiten una variedad de operaciones básicas, estas son agregar elementos, eliminar elementos, actualizar valores y comprobar si una clave esta presente en el diccionario.

* Añadir elementos en diccionario:

mi_diccionario["nueva_clave"] = nuevo_valor

* Eliminar elementos en diccionarios:

del mi_diccionario["clave_a_eliminar"]

* Actualizar valores en diccionarios:

mi_diccionario["clave_existente"] = nuevo_valor

* Comprobar si existe valor en diccionarios:

if "clave" in mi_diccionario:

hacer algo si la clave está presente

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

Hay dos funciones que se utiliza para ordenar en Python, **sort** y **sorted**

La función **sort** () se usa para ordenar la lista original

Sintaxis del metodo Sort()
List.sort()

Ejemplo:
myList = [2, 19, 8, 32, 5, 10]
print(myList)

myList.sort()
print(myList)

[2, 19, 8, 32, 5, 10]
[2, 5, 8, 10, 19, 32]

Como podemos ver la lista se ordeno con la función **sort**

Sort es un método aplicado a la lista, en este metodo el valor devuelto no tiene valor de retorno, pero ordena los objetos en la lista.

Ejemplo de como utilizar la funcion **Sort()**

```
list = ['Google', 'Twiter', 'Soptify', 'Facebook']
list.sort()
print ( "list : ", list)
```

Resultado: List : ['Facebook', 'Google', 'Twiter', 'Spotify']

* Genera lista descendente

```
#lista
vowels = ['e', 'a', 'u', 'o', 'i']

 # Descendente
vowels.sort(reverse=True)

 # Resultado de salida
 print ('Salida en orden descendente:', vocales) ['u', 'o', 'i', 'e', 'a']
```

sorted puede ordenar todos los objetos iterables.
La función **sorted** no modifica la secuencia original, crea una nueva secuencia ordenada basados en la secuencia original. **sorted** puede pasarlo como un argumento.

Sintaxis:
sorted(iterable_name, key, reverse = False)

```
numeros = [8, 2, 6, 10, 17]
sorted_numeros = sorted(numeros)
Print(sorted_numeros)
[2, 6, 8, 10, 17]
```

Ejemplo para ordenar de forma descendente:

```
li = [1, 6, 3, 89, 5]
li_new = sorted(li, reverse = True)
```

```
print("Given list is {}.".format(li))
print("Sorted list is {}.".format(li_new))
```

Salida: La lista dada es [1, 6, 3, 89, 5].
La lista ordenada es [89, 6, 5, 3, 1].

5. ¿Qué es un operador de reasignación?

El termino «reasignar» se utiliza cuando se asigna un nuevo y diferentes valores a una variable ya existente cada vez que sea necesario. Se utiliza para actualizar variables, cambiar el comportamiento de un programa.
Por ejemplo, cuando queremos modificar una tupla dado que son inmutables pode-mos aprovechar la reasignación y crear una nueva tupla y poder realizar todos los cambios en ese nuevo elemento.

Ejemplo de reasignacion de variables en Python:

```
mi_texto = ´Hola por aqui’
variable_mixta = ´Texto’
mi_numero = 242

#Reasignacion de valores
mi_texto = ´Hola por aqui’
variable_mixta = ´15.6’
mi_numero = 202502

nueva_variable = mi_numero

print(mi_texto)
print(mi_numero)
print(variable_mixta)
print(nueva_variable)
```

* Al empezar con un valor, y luego asignar otro, solo el último valor es consi derado y el valor anterior es sustituido totalmente, por lo tanto es una resigna cion de variables.

* Como se puede ver la variable_mixta inicio como una cadena de texto y ter mino siendo un numero decimal, Esto es posible gracias a que Python es muy flexible.

* La variable llamada variable_nueva no se le asigno un valor literal sino otra variable, por lo tanto nueva_variable paso a ser una replica de mi_numero ya que recibido el mismo valor.