Preguntas teóricas.

1. Diferencia entre lista y tupla.

Las dos entran en la categoría de colecciones de Python. Son agrupaciones de tipos de datos.

La lista es muy descriptiva. Suele ser una lista normal de artículos. Podemos poner cualquier colección de datos. Y todos ellos pueden ser de diferente valor, pueden ser números, una cadena u otra lista por ejemplo.

Las listas son mutables, se les pueden añadir objetos, moverlos, eliminarlos o llamarlos en cualquier momento.

Las listas en Python se guardan entre corchetes [] Cada cadena se mete dentro de los corchetes con comillas y cada una de ellas es un elemento único.

```
users =[‘Alba’, ‘Ibai’,]
```

Cuando imprimimos esta lista dejan de ser un conjunto de cadenas y se convierten en una estructura de datos real.

Aunque la tupla es muy similar a una lista la diferencia más importante es que los elementos dentro de ella son inmutables. No puedes añadir, eliminar, variar el orden o cambiar una cadena por otra (en las listas si). En una tupla tienes que actualizar en otra variable y reasignarla.

En las listas puedes usar funciones como “insert” para incorporar un nuevo elemento.

```
users.insert (0, ‘Zuri’)
print(users)      [‘Zuri’, ‘Alba’, ‘Ibai’]
```

Puedes usar también “append” para colocar una nueva cadena al final de tu lista.

```
users.append(‘Raquel’)
print(users)    [‘Zuri’, ‘Alba’, ‘Ibai’, ‘Raquel’]
```

Puedes consultar donde está colocado un elemento.

```
print(users[1])    Alba. La consulta te lo muestra como un objeto específico de esa lista.
print([users[1]])  [‘Alba’] . Así nos mostraría la consulta de este elemento pero dentro de la lista.
```

Puedes cambiar un elemento por otro.

```
users[1] = ‘hija’
print(users)   [‘Zuri’, ‘hija’, ‘Ibai’, ‘Raquel’]  . Las cadenas si son inmutables por lo tanto lo que hemos hecho aquí es reasignar un nombre.
```

Puedes eliminar elementos con funciones como “remove”, “pop” o “del”

```
users.remove(‘Ibai’)
print(users)  Elimina Ibai de la lista.  [‘Zuri’, ‘hija’, ‘Raquel’]
```

popped.users = users.pop() Elimina el último elemento de la lista Raquel. Lo borra de la lista pero puedes volver a rescatarlo si quieres usarlo en otro momento.

del users[2]

print(users) Elimina el elemento que esté colocado en el índice 2. 'Raquel'.

['Zuri', 'hija']

Todas estas funciones no se utilizan en las tuplas.

Aunque son muy similares si necesitas tener una estructura de datos que quieras cambiar tienes que utilizar una lista, pero si quieres mantenerla igual puedes usar una tupla.

Consultar los elementos guiándonos por su índice se puede hacer en las dos. Puedes acceder a los elementos cuando necesites. Para coger elementos, sacarlos o ejecutar cualquier cosa con ellos, en la tupla tienes que crear una nueva variable actualizada con los cambios. Y reasignarla a la tupla inicial que siempre se va a mantener igual. Los cambios se realizan en la nueva tupla.

La sintaxis de la tupla también es diferente, en vez de corchetes [para la lista] se utilizan paréntesis ().

tupla= ('mujer', 'edad', 'ciudad')

print(tupla) ('mujer', 'edad', 'ciudad')

tupla += ('pais' ,)

print(tupla) ('mujer', 'edad', 'ciudad', 'pais') Así podemos añadir elementos en la tupla. Reasignándolos en otra variable.

Cuando ejecutas un corte o un rango en una lista te devuelve la lista con los elementos cambiados. Cuando lo haces en una tupla te devuelve una tupla porque es muy intuitivo y mantiene la estructura de datos sin importar que trabajes con una lista o una tupla.

Las tuplas también las podemos convertir en listas para trabajar como si fueran una.

tupla= list(tupla)

2. Orden de las operaciones.

Las reglas para actuar en una serie matemática se rigen por el siguiente orden:

Please = Parans () Lo primero que hay que resolver es si tenemos algo entre paréntesis. (8+4)

Excuse = Exponents ** Lo segundo a resolver son los exponentes (8+4)**2

My = Multiplication *. Ahora buscamos si tenemos algo que multiplicar en nuestra operación.

Dear= División/ . Después tendríamos que buscar si hay algo para dividir.

Aunt= Addition + Ahora realizamos las sumas.

Sally= subtraction - . Lo último sería las restas.

Calculation =10–4 /2–(8+10)**2 *2

10-4/2-18**2*2

10-4/2-324*2

10-4/2-648

10-2-648

Print(calculation) -640.0

3. Diccionario.

   La sintaxis del diccionario es que usan llaves{} para colocar dentro de ellos los elementos y ponerlos en múltiples renglones.

Se empieza siempre con una cadena. Un diccionario almacena una variable que contiene un elemento (un valor) que a su vez lleva una llave.

betoño = {

“juv”: “Marcos”

}

Print(betoño) {‘juv’ : ‘Marcos’}

Para añadir más elementos, tantos como quieras después de nuestro primer valor ponemos una coma y seguimos en otro renglón.

betoño = {

“juv”: “Marcos”,

“inf”: “Hodei”,

“ale”: “Ibai”

}

Para buscar un elemento del diccionario, si sabemos el valor de índice, usamos el valor clave.

Pasamos el valor clave para acceder al valor (muy parecido a una lista)

alevin= betoño[‘ale']

print(alevin) Ibai

Se pueden hacer colecciones anidadas.

```
teams= {
        'betoño' : ['Marcos', 'Hodei', 'Ibai']
        'aurrera': ['Miguel', 'Eduardo']
        'alaves' : ['Ander', 'Xabi']
}
```

Esto es un diccionario en el que puedes realizar cualquier tipo de tareas. Agarrar elementos, agregar nuevos pares de valores, consultar elementos…

4. Diferencia entre el método ordenado y la función de ordenación.

Tenemos una lista que está ordenada solo por sus valores dependiendo de donde están colocados. Sort te ordena la lista alfabéticamente o de mayor a menor.

```
list= ['Ibai', 'Alba', 'Zuri']
```

list.sort() después de sort no vamos a dar ningún argumento. Esto es muy intuitivo, pone toda la lista en orden alfabético

```
print(list)  ['Alba', 'Ibai', 'Zuri'] . Coloca la lista en orden alfabético.
```

Si lo quieres al revés y revertir la lista

```
list.sort(reverse = True)
print(list)    ['Zuri', 'Ibai', 'Alba']
```

Con los números hace lo mismo. Los coloca de menor a mayor o al revés.

```
numbers= [200,100,300,1,2]
numbers.sort()
print(numbers)    [1, 2, 100,200,300]
```

Así cambiamos toda la estructura de estos elementos.

Sort es una función que se utiliza en las listas. Modifica la lista sobre la marcha. Cambia la lista. No creamos una nueva con los cambios (esto lo hace el método de ordenación)

La otra opción es que podemos pasar todo a otra variable y que ella realice el ordenamiento de la lista."Sorted" se comporta igual que "sort" pero permite almacenar el nuevo valor dentro de una variable nueva. No cambia los valores de lugar, así que la lista primera permanece intacta y se puede usar siempre que se quiera.

```
numbers= [200,100,300,1,2]

sorted_numbers= sorted(numbers)

print(sorted_numbers)   [1, 2, 100,200,300]
```

No se altera la lista primera, si vuelves a llamarla te dará lo mismo del principio.

```
print(numbers)      [200,100,300,1,2]
```

Igual que hace “post” puede cambiar la lista al revés.

```
sorted_numbers = sorted(numbers, reverse=True)

print(sorted_numbers)      [200,100,300,1,2]
```

Si necesitas cambiar una lista y no te importa cambiarla puedes usar sort. Si quieres cambiarla pero deseas mantener la original usarías sorted creas una nueva lista distinta a la original.

5. Operador de reasignación.

Cuando reasignamos un valor es como sobreescribir el valor de un elemento.

Sirve para modificar el valor de lo que tenemos.

```
Sentence= ‘mi gato está en el tejado’

Sentence_one= ‘mi perro’

Sentence_one= sentence.replace(‘gato’ , ‘perro’ )

Print(sentence_one)    nos daría: mi perro está en el tejado.
```

La variable primera es sentence y no la cambiamos nunca, se mantendrá igual si volvemos a llamarla. Cuando reasignamos un valor nuevo en la oración, cogemos ese elemento y le decimos que queremos esa misma variable (sentence) pero con nuestro cambio.

En ningún momento cambiamos la primera oración. Nos aprovechamos de ella para crear una nueva, reasignando algún elemento.

Las tuplas por ejemplo también son inmutables así que para actualizarlas (cambiar algo) hay que hacerlo como una cadena.

Reasignamos para poder crear una nueva tupla y los cambios se hacen en esta nueva tupla. Hay que alterar esto dinámicamente y para ello se utiliza la reasignación.

```
tupla= ('claseA', 'grupo de estudiantes', 'redacción de texto')

tupla += (‘conclusiones’ , )

print(tupla)

('claseA', 'grupo de estudiantes', 'redaccion de texto', 'conclusiones')
```

La coma después de conclusiones es muy importante porque sin ella Python entiende que solo quieres configurar un orden de operaciones. Si añades la coma Python entiende que es una tupla.

Nunca aparecerá ‘conclusiones’ incluído en la tupla inicial.