Las preguntas teóricas son:

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

La principal diferencia es que las tuplas son inmutables, es decir, que no se pueden cambiar ni actualizar una vez creadas. Mientras que las listas son mutables, permitiéndonos modificarlas (agregar, eliminar o cambiar elementos).

La sintaxis que las definen también son diferentes. () para las tuplas y [] para las listas.

Mientras que es fácil acceder y modificar elementos de listas, también existen algunas maneras de modificar una tupla o acceder a elementos de una tupla usando por ejemplo " .list() ".

Ejemplos:

```
lista = ["piña", "coco", "manzana"]
print("1 => ", lista)

tupla = ("naranja", "pomelo", "lima")
print("2 => ", tupla)

# modificar una lista
lista[2]="pera"
print("3 => ", lista)

# modificar tupla
tupla_2 = list(tupla)
tupla_2[0]= "sandía"
tupla = tuple(tupla_2)
print("4 => ", tupla)

# añadir elemento a lista
lista.extend(["fresa"])
print("5 => ", lista)

# añadir elemento a tupla
tupla += ("limón",)
print("6 => ", tupla)



1 =>  ['piña', 'coco', 'manzana']
2 =>  ('naranja', 'pomelo', 'lima')
3 =>  ['piña', 'coco', 'pera']
4 =>  ('sandía', 'pomelo', 'lima')
5 =>  ['piña', 'coco', 'pera', 'fresa']
6 =>  ('sandía', 'pomelo', 'lima', 'limón')
```

2. ¿Cuál es el orden de las operaciones?

El orden en el que se realiza las operaciones siempre es el mismo.

P E M D A S

Please -> Parans ()
Excuse -> Exponents **
My -> Multiplication *
Dear -> Division /
Aunt -> Addition +
Sally -> Subtraction -

Indicando que el paréntesis es lo primero que se opera, seguido exponentes, multiplicaciones, divisiones y por último, las sumas y las restas.

```
(2+3)*6-12/(2+2^2)
5*6-12/(2+4)
5*6-12/6
30-2
28
```

3. ¿Qué es un diccionario Python?

Un diccionario en Python es una colección mutable de elementos a los que se accede por una clave (key) concreta.

Su estructura básica está formada por claves (keys) y valores (values), los cuales están vinculados, y pueden almacenarse, añadirse, eliminarse y recuperarse posteriormente. Es decir, un diccionario es una colección ordenada modificable de pares clave-valor.

Asimismo, los valores dentro del diccionario pueden estar formados de un único elemento o varios como: un listado, una tulpa e incluso otro diccionario.

En Python reconoceremos que estamos ante un diccionario si los elementos están rodeados por {}.

Ejemplos:

```python
# ejemplo diccionario

comidas = {
   "fruta" : ["manzana", "pera", "naranja"],
   "verduras" : ["acelgas", "zanahorias","lechuga"],
   "carnes" : ["pollo", "cerdo", "vaca"],
   "otros": ["salsas", "especias","lacteos"],
}

print ("Diccionario comidas =>  ", comidas)
print("listado frutas que tiene el diccionario =>  ", comidas["fruta"])

# añadir valores al diccionario
comidas["pescados"] = ["salmón", "lenguado", "sardinas"]
print("Diccionario con nuevos valores => ", comidas)

# buscar valores
frutas_disponibles = comidas.get('frutas', 'No se encuentra')
print("frutas disponibles => ", frutas_disponibles)

# eliminar valores
del comidas["otros"][2]
eliminados = comidas.pop("otros", 'No se encuentra')
print("elementos eliminados =>  ", eliminados)
print("diccionario final =>  ", comidas)
```

```
Diccionario comidas =>  {'fruta': ['manzana', 'pera', 'naranja'], 'verduras': ['acelgas', 'zanahorias', 'lechuga'], 'carnes': ['pollo', 'cerdo', 'vaca'], 'otros': ['salsas', 'especias', 'lacteos']}

listado frutas que tiene el diccionario =>  ['manzana', 'pera', 'naranja']

Diccionario con nuevos valores => {'fruta': ['manzana', 'pera', 'naranja'], 'verduras': ['acelgas', 'zanahorias', 'lechuga'], 'carnes': ['pollo', 'cerdo', 'vaca'], 'otros': ['salsas', 'especias', 'lacteos'], 'pescados': ['salmón', 'lenguado', 'sardinas']}

frutas disponibles => ['manzana', 'pera', 'naranja']

elementos eliminados =>  ['salsas', 'especias']

diccionario final =>  {'fruta': ['manzana', 'pera', 'naranja'], 'verduras': ['acelgas', 'zanahorias', 'lechuga'], 'carnes': ['pollo', 'cerdo', 'vaca'], 'pescados': ['salmón', 'lenguado', 'sardinas']}
```

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

Ambos métodos ordenan alfanuméricamente una lista. La diferencia radica en que el método sort() lo hace sobre la propia lista inicial y la modificación es permanente, y el metodo sorted() nos devuelve una nueva lista ordenada, quedando la lista original sin cambios.

Por este motivo, otra de las diferencias es que al aplicarse los cambios de forma permanente a la lista original, Python no nos permite guardar sort() en una variable, mientras que con sorted() sí.

Ejemplo:

```
# método sort()
libros = ["misterio", "cocina", "romance", "terror", "aventuras", "suspense",
"infantiles"]
libros.sort(reverse=True)
print("libros =>  ", libros)
libros.sort()
print("libros =>  ", libros)

asistentes = [140, 650, 63,785, 334, 89, 237]
asistentes.sort(reverse=False)
print("asistentes =>  ", asistentes)


libros =>   ['terror', 'suspense', 'romance', 'misterio', 'infantiles', 'cocina',
'aventuras']
libros =>   ['aventuras', 'cocina', 'infantiles', 'misterio', 'romance', 'suspense',
'terror']
asistentes =>   [63, 89, 140, 237, 334, 650, 785]



# método sorted()

colores = ["rojo", "verde", "azul", "naranja", "violeta", "amarillo", "blanco", "rosa"]
colores_ordenados= sorted(colores)
print("colores oredenados => ", colores_ordenados)
print("colores =>  ", colores)


colores oredenados =>  ['amarillo', 'azul', 'blanco', 'naranja', 'rojo', 'rosa', 'verde',
'violeta']
colores =>   ['rojo', 'verde', 'azul', 'naranja', 'violeta', 'amarillo', 'blanco', 'rosa']
```

5. ¿Qué es un operador de reasignación?

Los operadores de asignación o *assignment operators* nos permiten realizar una operación y almacenar su resultado en la variable inicial.

Como estamos asignando un resultado a una variable, el operador de reasignación tomará la forma de (operador matemático **+,-,*,/,** ...) y un **=.**

```
# Assignment Operators

num = 100
print ("Número inicial =>  ", num)

num += 10    # num = num + 10
print ("numero anterior aplicando suma => ", num)

num -= 10   # num = num - 10
print ("numero anterior aplicando resta => ", num)

num *= 2    # num = num * 10
print ("numero anterior aplicando multiplicación => ", num)

num /= 10  # num = num / 10
print ("numero anterior aplicando división => ", num)

num **= 2
print ("numero anterior aplicando exponencial => ", num)

num //= (num * 1/9)
print ("numero anterior aplicando entero div => ", num)

num %= 2  # saber si número es par o no
print ("numero anterior -> resto division => ", num)


Número inicial =>   100
número anterior aplicando suma =>  110
número anterior aplicando resta =>  100
número anterior aplicando multiplicación =>  200
número anterior aplicando división =>  20.0
número anterior aplicando exponencial =>  400.0
número anterior aplicando entero div =>  9.0
número anterior -> resto division =>  1.0 (impar)
```