Independientemente del uso que se le de a las funciones *lambda* podemos decir que las mismas en una línea de código son capaces de resolver muy eficientemente problemas.

Si bien si sintaxis a simple vista no es complicada, lo cierto es que a medida que acomplejamos los requerimientos si que estas se vuelven algo complejas.
Y si bien esto último no es nada que chatGTP no arregle, no esta demás ejercitar la lógica y entender que estamos haciendo o en que nos ayudo chatGTP.
Para este repositorio de IA en particular no utilice chatGTP más que para crear ejercicios a medida que veía la documentación adjuntada tanto en la carpeta como en los readme.md, sin embargo con las funciones lambda no fue el caso, acá se puso bien complicada la cosa.
Analizaré las que considero que verdaderamente significaron un problema al cual no estuve, casi siempre por falta de conocimiento de las funciones y métodos de python, a la altura de poder resolverlos correctamente.

Explicación ejercicio 12:

12. Dada una lista de números, utiliza una función lambda junto con `filter()` para crear una nueva lista que contenga solo los números pares.

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(pares)  # Salida: [2, 4, 6, 8, 10]
```

La lista la creamos nosotros, la misma en este caso se llama *numeros*, luego creamos otra lista que guarde otra que los requerimientos en cuestión.
Debemos usar *list()* previo a *filter(filtro, estructura)* dado que sino el *type* de la nueva estructura será filter, y queremos que sea una lista.
La función lambda en este caso no se guarda en una variable como tal ya que solo la requerimos acá, por lo que filter busca los valores filtrando según si su resto es 0, es decir: es par, de esta manera retornando una lista usando la función filter.

Explicación ejercicio 13:

13. Crea una lista de tuplas donde cada tupla contenga un nombre y una edad. Utiliza una función lambda como clave para ordenar la l
por edad.

```
lista_nom_edad = [("Martin", 21),("Benjamin", 25),("Carlos", 18)]

orden = lambda lista : sorted(lista, key=lambda x: x[1])

lista_ordenada = orden(lista_nom_edad)

print(lista_ordenada)
```

La lista de tuplas la creamos nosotros.
En este caso si guardamos la función en una variable.
La función lambda debe tener como parámetro una lista, ordenaremos esta con la función ***sorted***, ordenaremos la lista que pasamos como parámetro teniendo en cuenta el valor guardado en el índice 1, en este caso la edad.
Para esto usaremos otra función lambda solo para acceder a este valor.
Recordemos que el primer índice es el 0.

Explicación ejercicio 14:

14. Dada una lista de números, utiliza `map()` y una función lambda para crear una nueva lista que contenga el doble de cada número.

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]

doble_numero = list(map(lambda x: x * 2, numeros))

print(doble_numero)
```

La función map sigue la siguiente estructura:

map(function, iterables)

Por un tema tanto de escritura en consola como por requerimientos de la consigna debemos retornar una lista, en la función pasaremos que los elementos de número, que Python ya sabe que son x, se multiplicarán con 2.
Viniendo de Java y C esto es más que confuso: ya sabe que x son los elementos de números, es una completa locura, pero es así.

Todo lo visto hasta ahora puede ser combinado de infinitas maneras posibles.
Por ejemplo: podemos tener una función ***map*** que aplique una determinada función sobre determinada estructura y que itere sobre una estructura que a su vez este filtrada según requerimientos.
Un ejemplo de esto se verá en la explicación del ejercicio 17.

Explicación ejercicio 17:

17. Crea una lista de números y utiliza `map()` y `filter()` en combinación con funciones lambda para obtener una lista de los cuadrados de los números impares.

```python
lista_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1000, 1001]

list_cuad_imp = list(map(lambda x: x ** 2, filter(lambda x: x%2 != 0, lista_numeros)))

print(f"La lista es {lista_numeros} y la lista con sus elementos impares al cuadrado es {list_cuad_imp}")
```

Debemos retornar una lista, por lo que utilizaremos list().

Luego usaremos el map() para pasar una función e iterar sobre la estructura pasada en el segundo parámetro, en este caso el cuadrado de los elementos será la función y, el segundo parámetro, la lista filtrada con los elementos impares de la lista_numeros.

Este ejercicio integra list, map y filter en una sola estructura y línea de código.

No es que este mal aplicar una función con una lógica con condicionales y estructuras de control que nos permitan manejar los requerimientos, sino que sin diferentes enfoques y técnicas.

Explicación ejercicio 18:

18. Dada una lista de nombres, utiliza una función lambda para crear un diccionario donde cada nombre sea una clave y su longitud sea e
valor.

```
lista_nombres = ["Martin", "Diego", "Alejandro", "Daniel", "Juan"]

diccionario_nombres_long = dict(map(lambda nombre: (nombre, len(nombre)), lista_nombres))

print(diccionario_nombres_long)
```

Debemos usar dict() para generar un diccionario, luego usaremos map para generar una lista que contenga una tupla con (nombre, longitud del nombre), siento nombre los elementos iterados en la lista_nombres

Explicación ejercicio 19:

19. Crea una lista de diccionarios donde cada diccionario represente una persona con claves `nombre` y `edad`. Usa `sorted()` junto con una función lambda para ordenar la lista por edad.

```
lista_diccionarios = [{"nombre":"Fulanito", "edad":80},
                      {"nombre":"Martin", "edad":21},
                      {"nombre":"Carlos", "edad":20},
                      {"nombre": "Alejo", "edad":30}]

lista_ordenada = sorted(lista_diccionarios, key=lambda x: x["edad"])
```

Debemos ordenar la lista tomando en cuenta la edad, para esto crearemos el indice x y ordenará mediante sorted por "edad".