preguntas del taller de práctica py

1. Creación y manipulación básica de listas

¿Por qué en la solución se utiliza `append()` en lugar de `insert()` para agregar los elementos ingresados por el usuario a la lista `frutas`?

RTA: Se usa `append()` porque agrega elementos al final de la lista de forma eficiente. `insert()` es útil para insertar en posiciones específicas, pero es menos eficiente si se usa repetidamente porque requiere mover elementos dentro de la lista.

---

2. Extendiendo una lista

¿Cuál es la diferencia fundamental entre usar `append()` e `extend()` para agregar varios elementos a la lista?

RTA: `append()` agrega un solo elemento (que puede ser una lista), mientras que `extend()` añade los elementos de otra lista individualmente, expandiendo la lista original.

---

3. Insertar elementos en posiciones específicas

En la solución, se pide al usuario un nuevo lenguaje y se coloca en la posición 1 con `insert(1, nuevo_lenguaje)`. ¿Qué sucedería si se usara `append()` en lugar de `insert()` en ese contexto?

RTA: `insert(1, nuevo_lenguaje)` coloca el elemento en la segunda posición. Si se usara `append()`, el nuevo lenguaje se agregaría al final en lugar de la posición deseada.

---

4. Eliminando elementos de una lista

¿Por qué antes de usar `remove()` para eliminar un animal de la lista, en la solución se verifica con `if animal in animales:`? Explica la importancia de esta verificación.

RTA: `remove()` lanza un error si el elemento no está en la lista. Verificar con `if animal in animales:` evita este error y permite manejar la situación de manera controlada

---

5. Extrayendo elementos con `pop()`

En la solución, se usan tanto `pop()` sin índice como `pop(indice)`. ¿En qué casos conviene usar `pop()` sin índice y en cuáles `pop(indice)`?
RTA: `pop()` sin índice elimina y devuelve el último elemento, útil cuando el orden no es importante. `pop(indice)` extrae un elemento específico, útil cuando se requiere una posición en particular.

---

6. Limpiando una lista con `clear()`

¿Por qué crear una función `limpiar_lista(lista)` que llama a `clear()` puede ser útil en un programa más grande? Explica un escenario donde se necesite esta modularidad.
RTA: Facilita la reutilización del código. En un programa más grande, podría usarse en un sistema de gestión donde se necesite resetear datos antes de una nueva operación.

---

7. Encontrar el índice de un elemento con `index()`

La solución sugiere el uso de `try/except` al aplicar `lista.index(elemento)`. ¿Qué error se evita y por qué es conveniente capturarlo con `try/except`?
RTA: Si el elemento no está en la lista, `index()` lanza un error `ValueError`. Usar `try/except` previene la interrupción inesperada del programa.

---

8. Contar ocurrencias con `count()`

Si `lista.count(elemento)` devuelve 0, la solución muestra un mensaje de 'no encontrado'. ¿Por qué es más eficiente usar `count()` en vez de recorrer toda la lista con un `for` para contar manualmente?
RTA: `count()` es más eficiente porque está optimizado internamente en Python y evita recorrer manualmente la lista con un `for`.

---

9. Ordenar una lista con `sort()`

¿Cómo se implementa en la solución la ordenación descendente utilizando `sort()`? Explica brevemente qué parámetro del método `sort()` se modifica.
RTA: Se usa `sort(reverse=True)`, lo que invierte el orden ascendente predeterminado.

---

10. Invertir una lista con `reverse()`

En la solución, después de usar `reverse()`, la lista queda invertida de forma permanente. ¿Qué alternativa existe si solo quisieras obtener una versión invertida sin modificar la lista original?
RTA: Se puede usar `lista[::-1]`, que devuelve una copia invertida sin alterar la original.

---

11. Copiar listas con `copy()`

En la solución se compara `lista_copia = lista_original.copy()` con `lista_copia = lista_original`. Explica la diferencia en el comportamiento de ambas asignaciones.
RTA: `copy()` crea una nueva lista independiente. Con `lista_copia = lista_original`, ambas variables apuntan a la misma lista, por lo que modificar una afecta a la otra.

---

12. Creación de una agenda de contactos

¿Por qué la solución propone almacenar cada contacto como un diccionario (con llaves como `nombre` y `tel`) dentro de la lista, en lugar de usar solo listas anidadas o tuplas?
RTA: Los diccionarios permiten acceder a los datos por nombre en lugar de usar índices, facilitando la manipulación y búsqueda.

---

13. Función para mostrar contactos con filtro

En el filtrado de contactos, se usa la expresión `if filtro.lower() in contacto["nombre"].lower()`. Explica por qué se aplican los métodos `lower()` a ambas cadenas.

RTA: Permite hacer la búsqueda sin importar mayúsculas o minúsculas, evitando que "Carlos" y "carlos" sean tratados como diferentes.

---

14. Clasificación de números (uso de `if` y ciclos)

En la solución de `clasificar_numeros`, se recorre la lista para separar pares e impares. ¿Podrías mencionar una forma diferente (con otra estructura o función) de lograr la misma clasificación en menos líneas?
RTA: Se puede usar listas
pares = [n for n in numeros if n % 2 == 0]
impares = [n for n in numeros if n % 2 != 0]

---

15. Menú con `match` (versión de Python 3.10+)

La solución muestra un ejemplo de uso de `match opcion:`. Menciona una ventaja de `match` sobre los `if-elif-else` tradicionales a la hora de manejar múltiples casos.
RTA: `match` es más legible y eficiente para múltiples casos, evitando evaluaciones repetitivas.

---

16. Menú más complejo con `match` y listas

En el menú de tareas, cuando se elige la opción '3. Eliminar tarea', la solución usa `remove()`. ¿Qué pasa si se intenta eliminar una tarea que no existe en la lista y no se maneja esa excepción?
RTA: Si la tarea no existe en la lista, `remove()` generará un error `ValueError`. Se debe verificar con `if tarea in lista:` antes de eliminar.

---

17. Uso de `if` y `match` en la misma función

En la función `evaluar_estado(nota)`, primero se valida con `if` que la nota esté entre 0 y 10, y luego se usa `match`. ¿Por qué es importante esta verificación previa a `match`?
RTA: `match` no maneja rangos de valores fácilmente. Validar primero evita errores si se ingresan valores fuera del esperado.

---

18. Búsqueda dentro de una lista (uso de `for`, `if` y funciones de cadena)

La función `buscar_personas(lista, texto_buscar)` recorre cada nombre y verifica `if texto_buscar.lower() in nombre.lower()`. ¿Qué sucede si en lugar de `in` se usara un operador de igualdad (`==`)?
RTA: `in` verifica si un texto está contenido en otro. `==` exige coincidencia exacta, lo que haría que solo nombres idénticos sean encontrados.

---

19. Conversión de cadena a lista y viceversa

En la solución se hace `frase.split()` y luego `" ".join(lista_de_palabras)`. Explica qué representa el espacio `" "` en el método `join()`.
RTA: Especifica que las palabras se unirán con un espacio entre ellas, formando una frase coherente.

---

20. Filtros con condicionales y creación de sublistas

En la solución de `filtrar_lista(numeros, tipo)`, se crean nuevas listas para pares o impares usando un ciclo `for`. ¿Por qué no se modifican los elementos directamente en la lista principal?
RTA: Se evita alterar los datos originales, permitiendo reutilizar la lista en otros contextos.

---

21. Uso de `while` para validar entrada

La solución pide 3 números pares al usuario, validando con un `while`. ¿Qué pasaría si se coloca la condición del `while` al inicio sin controlar cuántos números se han ingresado ya? (Por ejemplo, `while numero % 2 != 0:`)
RTA: Se entraría en un bucle infinito si el usuario sigue ingresando números impares, ya que la condición no verifica cuántos pares han sido introducidos.

---

22. Creación de funciones con parámetros y retorno

En la solución de la función `sumar(a, b)`, se demuestra cómo retornar un valor. ¿En qué situaciones es más conveniente usar un retorno en lugar de imprimir directamente el resultado dentro de la función?

RTA: Retornar permite usar el resultado en otros cálculos o almacenarlo, mientras que imprimir solo lo muestra en pantalla.

---

23. Uso de `break` y `continue` en un ciclo

En la solución, se salta la impresión cuando el número es 5 y se rompe el ciclo cuando el número es 8. ¿Qué diferencia hay entre `break` y `continue` en términos de flujo del programa?
RTA: `break` detiene completamente el ciclo, mientras que `continue` salta a la siguiente iteración sin ejecutar el resto del código en esa vuelta.

---

24. Ordenamiento de lista de cadenas con `sort()` y uso de `key`

La solución muestra `sort(key=len)` para ordenar por longitud de las cadenas. ¿Qué efecto tiene este parámetro `key=len` en el proceso de ordenamiento?
RTA: Ordena las cadenas por longitud en lugar de alfabéticamente.

---

25. Ejercicio integral de listas y cadenas

La solución convierte cada palabra a mayúsculas con `upper()` antes de agregarla a la lista. ¿Cómo afectaría al resultado final si se hiciera después de ordenar la lista?
RTA: Si se usa antes, todas las palabras estarán en mayúsculas antes de ordenarse. Si se usa después, el ordenamiento puede verse afectado porque Python distingue entre mayúsculas y minúsculas.

---

26. Función que reciba lista y devuelva estadísticas

En la solución de `estadisticas(numeros)`, se retorna `(maximo, minimo, promedio)`. ¿Por qué usar una tupla para retornar múltiples valores es preferible a retornar una lista o usar variables globales?
RTA: Son inmutables y permiten devolver múltiples datos sin modificar variables globales.

---

27. Programa de calificaciones con decisión

En la solución, se usa un ciclo que termina al ingresar un valor negativo. ¿Qué ventaja tiene este enfoque frente a pedirle al usuario de antemano cuántas calificaciones desea ingresar?
RTA: Es más flexible porque el usuario decide cuándo detenerse sin especificar un número fijo de calificaciones.

---

28. Combinar dos listas con `extend()` de forma controlada

En la solución, se confirma con el usuario (sí/no) antes de combinar las listas. ¿Qué aspectos de usabilidad o de seguridad se cubren con este paso de confirmación?
RTA: Evita combinaciones accidentales que podrían ser difíciles de deshacer.

---

29. Buscador de índices múltiple

La solución recorre la lista para encontrar todos los índices donde aparece la palabra buscada. ¿Por qué no se usa directamente `index()` en un ciclo para este fin?
RTA: `index()` solo devuelve la primera aparición, por lo que requeriría ajustes adicionales para encontrar todas.

---

30. Función recursiva que sume elementos de una lista

En la solución, si la lista está vacía se retorna 0, de lo contrario se suma el primer elemento y se llama recursivamente con el resto. ¿Por qué es importante definir claramente el caso base en una función recursiva?
RTA: Sin un caso base, la función entraría en un ciclo infinito y provocaría un error de desbordamiento de pila (`RecursionError`).