# Lista 02: Estructuras de Datos

Has demostrado una buena comprensión de las estructuras de datos en Python y ha aplicado correctamente listas, diccionarios, tuplas y conjuntos. Sin embargo, hay áreas en las que podría refinar su código, como el manejo de bucles y la legibilidad.

* **Cosas positivas**:
  + Uso adecuado de estructuras de datos según el problema.
  + Buena implementación de operaciones como transposición de matrices y cálculo de divisores.
  + Claridad en la estructuración de las soluciones.
* **Posibles mejoras**:
  + Evitar bucles redundantes o mal estructurados, como el uso excesivo de bucles infinitos.
  + Simplificar algunas operaciones utilizando métodos predefinidos de Python.
  + Comentar más detalladamente el código para hacerlo más legible y comprensible.

### Análisis Ejercicio por Ejercicio

#### Ejercicio 1

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**:



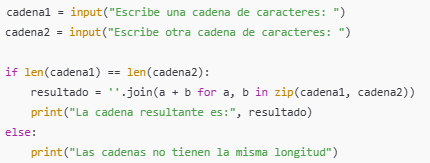
**Comentario**: Usar slicing ([::-1]) simplifica y optimiza la inversión de cadenas.

#### Ejercicio 2

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**:



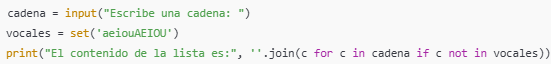
**Comentario**: Usar zip mejora la claridad al intercalar los caracteres de ambas cadenas.

#### Ejercicio 3

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**:

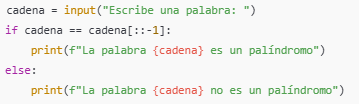
**Comentario**: La solución original funciona bien, pero set mejora la eficiencia en búsquedas.

#### Ejercicio 4

**Estado del ejercicio**: Incorrecto.

**Errores encontrados**: No se está comprobando correctamente el palíndromo al usar listas innecesarias.

**Código optimizado**:



**Comentario**: La comparación directa con slicing es más eficiente y clara.

#### Ejercicio 5

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**: No es necesario.

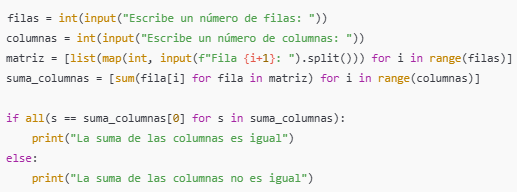
**Comentario**: Excelente implementación del contador para llenar una matriz con valores consecutivos.

#### Ejercicio 6

**Estado del ejercicio**: Incorrecto.

**Errores encontrados**: La lógica de comparación de las columnas no está implementada correctamente.

**Código optimizado**:

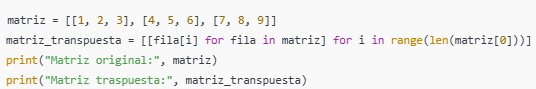
**Comentario**: Es importante estructurar bien la lógica para sumar columnas y comparar valores.

#### Ejercicio 7

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**:

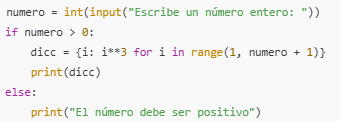
**Comentario**: Usar comprensión de listas para transponer matrices mejora la claridad del código.

#### Ejercicio 8

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**:



**Comentario**: Usar comprensión de diccionarios hace el código más compacto y eficiente.

#### Ejercicio 9

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**: No es necesario.

**Comentario**: La solución es clara y funcional.

#### Ejercicio 10

**Estado del ejercicio**: Correcto.

**Errores encontrados**: Ninguno.

**Código optimizado**: No es necesario.

**Comentario**: El bucle while y el cálculo del total están bien estructurados.

#### Ejercicio 11

**Estado del ejercicio:** Correcto.

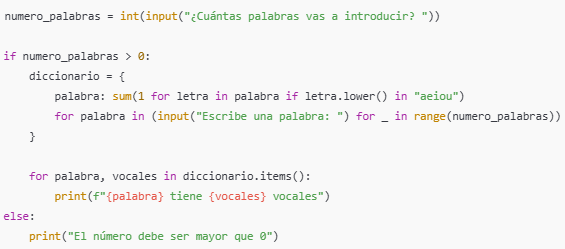
**Errores encontrados:** Uso de un bucle while True en el diseño principal. Aunque funcional, este enfoque no es una buena práctica de programación ya que puede dificultar la legibilidad y el control del flujo del programa.

**Código optimizado:** Aquí tienes una versión mejorada que elimina el while True y utiliza un enfoque más estructurado para evitar bucles infinitos:

**Comentario:** El programa cumple correctamente con los requisitos de recopilar, almacenar y procesar las notas de los estudiantes. Se recomienda evitar el uso de while True para mejorar la claridad del código y facilitar el mantenimiento. La versión optimizada estructura mejor el flujo del programa y maneja el caso en el que un estudiante podría no tener notas registradas.

### Ejercicio 12

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:  
El código puede simplificarse ligeramente al usar comprensión de listas:

**Comentario**: Usar comprensión de listas hace el código más compacto y legible.

### Ejercicio 13

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario. El código es claro y funcional.  
**Comentario**: Excelente uso de diccionarios para asociar números con sus divisores. Muy buena lógica.

### Ejercicio 14

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: Para encontrar el mínimo del conjunto de forma más directa:



**Comentario**: Utilizar la función min() es más eficiente y mejora la claridad.

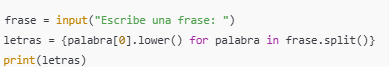
### Ejercicio 15

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:  
Puedes usar max() con key=ord para encontrar el máximo de forma más directa:

**Comentario**: Muy bien logrado. Esta optimización elimina la necesidad de realizar cálculos manuales.

### Ejercicio 16

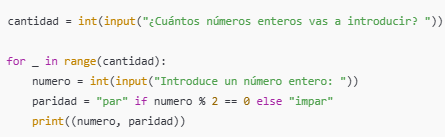
**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:  
El código ya es funcional. Sin embargo, puedes utilizar una expresión generadora para mayor claridad:



**Comentario**: Usar un conjunto (set) es una buena elección para almacenar las primeras letras únicas.

### Ejercicio 17

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:Usar una expresión condicional en una sola línea para determinar la paridad mejora la claridad:



**Comentario**: Muy bien implementado. La optimización ayuda a simplificar la estructura.

### Ejercicio 18

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario.  
**Comentario**: Excelente uso de comprensión de listas. La solución es clara y eficiente.

### Ejercicio 19

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario.  
**Comentario**: Muy bien estructurado. Usar tuple() y unpacking es una práctica avanzada y muy útil.

### Ejercicio 20

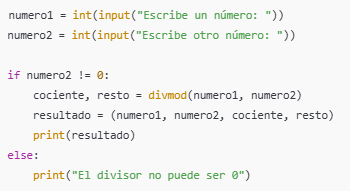
**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:El uso de comprensión de diccionarios mejora la legibilidad:



**Comentario**: Muy buen uso de listas y diccionarios para resolver el ejercicio.

### Ejercicio 21

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: El uso de divmod simplifica la obtención del cociente y resto:



**Comentario**: El código es funcional, pero esta optimización hace que sea más legible.

### Ejercicio 22

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:Puedes añadir validaciones para asegurar entradas válidas:



**Comentario**: Excelente trabajo al manejar la conversión de grados a radianes. Considera validar entradas no numéricas para mayor robustez.