# Lista 10: Numpy

En esta lista, se demuestra un manejo competente de **NumPy**, aplicando sus funciones y métodos para resolver problemas relacionados con matrices, arreglos y transformaciones de datos. Las soluciones muestran una comprensión básica de la librería, aunque algunas respuestas pueden beneficiarse de una mayor precisión y optimización. Has demostrado una buena comprensión de NumPy y tu capacidad para resolver problemas prácticos. Aunque las soluciones cumplen con los requisitos, hay margen para mejorar en la optimización del código y el uso de funciones específicas de NumPy.

* **Cosas positivas**:
  + Uso correcto de funciones esenciales como reshape, concatenate, y frompyfunc.
  + Implementación de estructuras de control y métodos adecuados para trabajar con arreglos.
  + Creatividad en las soluciones de minijuegos y en el diseño de funciones personalizadas.
* **Posibles mejoras**:
  + Evitar bucles innecesarios cuando NumPy puede realizar operaciones vectorizadas más eficientemente.
  + Comentar más detalladamente el código para explicar la lógica detrás de cada solución.
  + Corregir algunos errores menores y mejorar el manejo de excepciones en casos de entrada no válida.

### Análisis Ejercicio por Ejercicio

#### Ejercicio 1

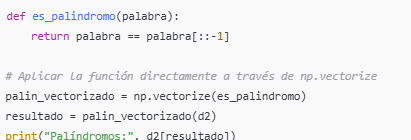
**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario.  
**Comentario**: Buena solución. Utilizas reshape correctamente para transformar el arreglo a 3D. Asegúrate de siempre verificar las dimensiones antes de aplicar esta función.

#### Ejercicio 2

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario.  
**Comentario**: Has usado concatenate de forma eficiente para combinar los arreglos. El resultado cumple con las especificaciones del problema.

#### Ejercicio 3

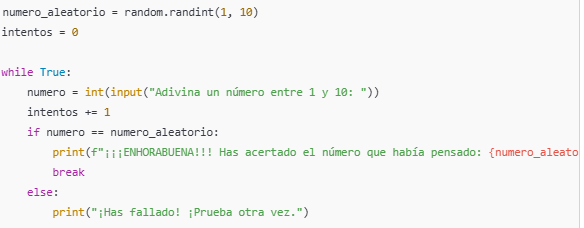
**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:



**Comentario**: La solución es adecuada, pero usando np.vectorize en lugar de frompyfunc, el código es más conciso y eficiente.

#### Ejercicio 4

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Uso de un bucle infinito.  
**Código optimizado**:

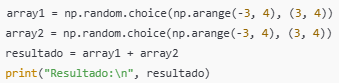
**Comentario**: El código se puede simplificar y hacer más claro manteniendo una lógica de salida bien definida para evitar bucles infinitos.

#### Ejercicio 5

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: La solución ya es adecuada y bien estructurada.  
**Comentario**: Excelente implementación del minijuego. Manejas correctamente las comparaciones de arreglos usando las funciones de NumPy.

#### Ejercicio 6

**Estado del ejercicio**: Incorrecto.  
**Errores encontrados**: Uso inapropiado de random.choices, lo que genera errores en matrices multidimensionales.  
**Código optimizado**:



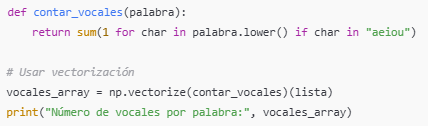
**Comentario**: Asegúrate de utilizar métodos específicos de NumPy como np.random.choice para evitar errores y garantizar la correcta generación de matrices.

#### Ejercicio 7 (Opcional)

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**: No es necesario.  
**Comentario**: La división del arreglo en subarreglos cumple con el objetivo del ejercicio. Podrías usar np.split para una solución más elegante.

#### Ejercicio 8 (Opcional)

**Estado del ejercicio**: Correcto.  
**Errores encontrados**: Ninguno.  
**Código optimizado**:



**Comentario**: Utilizando np.vectorize o comprensiones de lista puedes optimizar la solución y hacerla más legible.