Línea 1: import numpy as np

- Importa la biblioteca `numpy` y le asigna el alias `np`. Esta biblioteca es fundamental para operaciones numéricas, especialmente con arrays multidimensionales, que se usarán para definir los vértices del cubo y realizar transformaciones matriciales.

Línea 2: import matplotlib

- Importa la biblioteca principal `matplotlib`, utilizada para la creación de gráficos y visualizaciones en Python.

Línea 3: matplotlib.use('TkAgg') # Si usas Jupyter, puedes omitir esta línea y utilizar %matplotlib notebook

- Llama al método `use` del módulo `matplotlib` para especificar el backend gráfico a utilizar, en este caso `'TkAgg'`. El backend es el responsable de renderizar los gráficos en una ventana o interfaz. 'TkAgg' utiliza el toolkit Tkinter.

- El comentario adjunto aclara que esta línea podría ser innecesaria en entornos como Jupyter Notebook, donde se pueden usar comandos "mágicos" como `%matplotlib notebook` para habilitar la interactividad directamente en el notebook.

Línea 4: import matplotlib.pyplot as plt

- Importa el submódulo `pyplot` de la biblioteca `matplotlib` y le asigna el alias `plt`. `pyplot` proporciona una interfaz sencilla, similar a la de MATLAB, para crear figuras, ejes y realizar trazados.

Línea 5: from mpl_toolkits.mplot3d.art3d import Poly3DCollection

- Importa específicamente la clase `Poly3DCollection` desde el módulo `art3d`, que forma parte del toolkit `mplot3d` de Matplotlib. Esta clase es esencial para dibujar colecciones de polígonos en un gráfico 3D, como las caras de un cubo.

Línea 6: from matplotlib.animation import FuncAnimation

- Importa específicamente la clase `FuncAnimation` desde el módulo `animation` de Matplotlib. Esta clase permite crear animaciones actualizando una figura en intervalos regulares mediante la llamada repetida a una función definida por el usuario.

Línea 7: (En blanco)

- Línea en blanco: Separa el bloque de importaciones de la definición de la primera función, mejorando la legibilidad del código.

Línea 8: def crear_cubo():

- Define una función llamada `crear_cubo` que no acepta ningún parámetro.

Línea 9:  """

- Inicio del docstring (cadena de documentación) de la función `crear_cubo`. Los docstrings se usan para documentar lo que hace una función, sus parámetros y lo que devuelve.

Línea 10:  Crea un cubo centrado en el origen con lados de longitud 2.

- Línea del docstring: Describe el propósito principal de la función: generar la geometría de un cubo unitario (lados de -1 a 1, longitud 2) centrado en el punto (0,0,0).

Línea 11:  Devuelve:

- Línea del docstring: Indica que a continuación se describirán los valores que retorna la función.

Línea 12:  vertices: array con las coordenadas (x, y, z) de cada vértice.

- Línea del docstring: Describe el primer valor de retorno: un array de NumPy llamado `vertices`. Este array contendrá las coordenadas tridimensionales (x, y, z) de los 8 vértices que definen el cubo.

Línea 13:  caras: lista de caras del cubo, donde cada cara es una lista de vértices.

- Línea del docstring: Describe el segundo valor de retorno: una lista llamada `caras`. Cada elemento de esta lista representará una cara del cubo y será, a su vez, una lista que contiene las coordenadas de los vértices que forman esa cara.

Línea 14:  """

- Fin del docstring de la función `crear_cubo`.

Línea 15:  vertices = np.array([[-1, -1, -1],

- Dentro de la función `crear_cubo`: Comienza la definición de la variable `vertices`. Se le asigna un array NumPy creado con `np.array()`.

- Esta línea define las coordenadas `(x, y, z)` del primer vértice del cubo: `[-1, -1, -1]`.

Línea 16:  [ 1, -1, -1],

- Define las coordenadas del segundo vértice del cubo: `[ 1, -1, -1]`. La indentación adicional ayuda a alinear visualmente las coordenadas de los vértices.

Línea 17:  [ 1,  1, -1],

- Define las coordenadas del tercer vértice del cubo: `[ 1,  1, -1]`.

Línea 18:  [-1,  1, -1],

- Define las coordenadas del cuarto vértice del cubo: `[-1,  1, -1]`.

Línea 19:  [-1, -1,  1],

- Define las coordenadas del quinto vértice del cubo: `[-1, -1,  1]`.

Línea 20:  [ 1, -1,  1],

- Define las coordenadas del sexto vértice del cubo: `[ 1, -1,  1]`.

Línea 21:  [ 1,  1,  1],

- Define las coordenadas del séptimo vértice del cubo: `[ 1,  1,  1]`.

Línea 22:  [-1,  1,  1]])

- Define las coordenadas del octavo y último vértice del cubo: `[-1,  1,  1]`.

- El `]])` cierra la lista de listas que define los vértices y la llamada a la función `np.array()`. Ahora `vertices` es un array NumPy de 8x3.

Línea 23: (En blanco)

- Dentro de la función `crear_cubo`: Línea en blanco para separar visualmente la definición de `vertices` de la definición de `caras`.

Línea 24:  caras = [ [vertices[j] for j in [0, 1, 2, 3]],  # Cara trasera

- Dentro de la función `crear_cubo`: Comienza la definición de la variable `caras`, que será una lista.

- El primer elemento de la lista `caras` se crea usando una lista por comprensión (list comprehension). Esta comprensión itera sobre los índices `[0, 1, 2, 3]` y para cada índice `j`, recupera el vértice correspondiente `vertices[j]` del array `vertices`. El resultado es una lista de 4 arrays (los vértices de la cara).

- El comentario `# Cara trasera` indica qué cara del cubo representa esta lista de vértices.

Línea 25:  [vertices[j] for j in [4, 5, 6, 7]],  # Cara frontal

- Define el segundo elemento de la lista `caras` usando otra lista por comprensión. Selecciona los vértices con índices 4, 5, 6 y 7 para formar la cara frontal.

- El comentario `# Cara frontal` identifica esta cara.

Línea 26: [vertices[j] for j in [0, 1, 5, 4]], # Cara inferior

- Define el tercer elemento de la lista `caras` seleccionando los vértices con índices 0, 1, 5 y 4 para formar la cara inferior.

- El comentario `# Cara inferior` identifica esta cara.

Línea 27: [vertices[j] for j in [2, 3, 7, 6]], # Cara superior

- Define el cuarto elemento de la lista `caras` seleccionando los vértices con índices 2, 3, 7 y 6 para formar la cara superior.

- El comentario `# Cara superior` identifica esta cara.

Línea 28: [vertices[j] for j in [1, 2, 6, 5]], # Cara derecha

- Define el quinto elemento de la lista `caras` seleccionando los vértices con índices 1, 2, 6 y 5 para formar la cara derecha.

- El comentario `# Cara derecha` identifica esta cara.

Línea 29: [vertices[j] for j in [4, 7, 3, 0]] ] # Cara izquierda

- Define el sexto y último elemento de la lista `caras` seleccionando los vértices con índices 4, 7, 3 y 0 para formar la cara izquierda.

- El `]` cierra la lista `caras`.

- El comentario `# Cara izquierda` identifica esta cara.

Línea 30: return vertices, caras

- Dentro de la función `crear_cubo`: La sentencia `return` devuelve los dos valores generados: el array `vertices` y la lista `caras`. Se devuelven como una tupla.

Línea 31: (En blanco)

- Línea en blanco: Separa la definición de la función `crear_cubo` de la siguiente función `sesgar_x_z`.

Línea 32: def sesgar_x_z(vertices, shear_factor):

- Define una función llamada `sesgar_x_z` que acepta dos parámetros: `vertices` (que se espera sea un array NumPy como el devuelto por `crear_cubo`) y `shear_factor` (un número que determinará la intensidad del sesgado).

Línea 33:  """

- Inicio del docstring de la función `sesgar_x_z`.

Línea 34:  Aplica una transformación de sesgado al cubo.

- Línea del docstring: Explica que la función aplica una transformación geométrica de sesgado (shear).

Línea 35: (En blanco)

- Línea del docstring: Línea en blanco para separación visual dentro del docstring.

Línea 36:  Se sesga el eje X en función de la coordenada Z:

- Línea del docstring: Especifica el tipo de sesgado: la coordenada X de cada punto se modificará basándose en su coordenada Z original.

Línea 37:  x' = x + shear_factor * z

- Línea del docstring: Muestra la fórmula matemática para calcular la nueva coordenada x (`x'`) a partir de la coordenada x original (`x`), la coordenada z original (`z`) y el `shear_factor`.

Línea 38:  y' = y

- Línea del docstring: Muestra la fórmula para la nueva coordenada y (`y'`), que permanece igual a la original (`y`).

Línea 39:  z' = z

- Línea del docstring: Muestra la fórmula para la nueva coordenada z (`z'`), que también permanece igual a la original (`z`).

Línea 40: (En blanco)

- Línea del docstring: Línea en blanco para separación visual.

Línea 41:  Parámetros:

- Línea del docstring: Indica el inicio de la descripción de los parámetros de la función.

Línea 42:  vertices: array de vértices (n, 3).

- Línea del docstring: Describe el parámetro `vertices`, indicando que debe ser un array (presumiblemente NumPy) con `n` filas (una por vértice) y 3 columnas (coordenadas x, y, z).

Línea 43:  shear_factor: factor de sesgado.

- Línea del docstring: Describe el parámetro `shear_factor` como el factor numérico que controla la magnitud del sesgado.

Línea 44:  Retorna:

- Línea del docstring: Indica el inicio de la descripción del valor de retorno.

Línea 45:  vertices_sesgados: array con los vértices transformados.

- Línea del docstring: Describe el valor que devuelve la función: un nuevo array NumPy que contiene las coordenadas de los vértices después de aplicarles la transformación de sesgado.

Línea 46:  """

- Fin del docstring de la función `sesgar_x_z`.

Línea 47:  # Matriz de sesgado para transformar X en función de Z.

- Dentro de la función `sesgar_x_z`: Un comentario que explica que el código siguiente define la matriz matemática que representa la transformación de sesgado descrita.

Línea 48:  shear_matrix = np.array([[1, 0, shear_factor],

- Dentro de la función `sesgar_x_z`: Comienza la creación de un array NumPy llamado `shear_matrix`. Esta matriz 3x3 representará la transformación lineal.

- Define la primera fila de la matriz: `[1, 0, shear_factor]`. Esto corresponde a la ecuación `x' = 1*x + 0*y + shear_factor*z`.

Línea 49:  [0, 1, 0],

- Define la segunda fila de la matriz: `[0, 1, 0]`. Esto corresponde a la ecuación `y' = 0*x + 1*y + 0*z = y`.

Línea 50:  [0, 0, 1]])

- Define la tercera fila de la matriz: `[0, 0, 1]`. Esto corresponde a la ecuación `z' = 0*x + 0*y + 1*z = z`.

- El `]])` cierra la definición del array `shear_matrix`.

Línea 51:  return vertices.dot(shear_matrix.T)

- Dentro de la función `sesgar_x_z`: Retorna el resultado de aplicar la transformación a los vértices.

- `shear_matrix.T` calcula la transpuesta de la matriz de sesgado.

- `vertices.dot(...)` realiza la multiplicación matricial entre el array de vértices (Nx3) y la matriz de transformación transpuesta (3x3). El resultado es un nuevo array (Nx3) con las coordenadas de los vértices transformados. La transpuesta se usa aquí porque los vértices están almacenados como filas.

Línea 52: (En blanco)

- Línea en blanco: Separa la definición de la función `sesgar_x_z` del bloque principal de código que configura y ejecuta la visualización.

Línea 53: # Configuración de la figura y el eje 3D

- Comentario que indica que las siguientes líneas se encargarán de preparar el entorno de Matplotlib para el gráfico 3D.

Línea 54: fig = plt.figure()

- Llama a `plt.figure()` para crear un nuevo objeto `Figure` de Matplotlib, que actúa como el contenedor principal para todos los elementos del gráfico (ejes, títulos, etc.). El objeto figura se asigna a la variable `fig`.

Línea 55: ax = fig.add_subplot(111, projection='3d')

- Llama al método `add_subplot` del objeto `fig` para añadir un conjunto de ejes (un subplot) a la figura.

- `111` es una notación abreviada para `(1, 1, 1)`, que significa: crear una cuadrícula de 1x1 subplots y seleccionar el primero.

- `projection='3d'` es el argumento clave que especifica que estos ejes deben ser capaces de manejar gráficos en 3 dimensiones.

- El objeto `Axes3D` resultante (los ejes 3D) se asigna a la variable `ax`.

Línea 56: (En blanco)

- Línea en blanco: Separa la creación de la figura/ejes de la creación de la geometría inicial del cubo.

Línea 57: # Crear el cubo inicial

- Comentario que indica que el siguiente bloque de código generará los datos del cubo y los preparará para ser dibujados.

Línea 58: vertices, caras = crear_cubo()

- Llama a la función `crear_cubo()` definida anteriormente.

- La tupla que devuelve la función (el array de vértices y la lista de caras) se desempaqueta, asignando el primer elemento a la variable `vertices` y el segundo a la variable `caras`.

Línea 59: poly = Poly3DCollection(caras, facecolors='lightcoral', edgecolors='black', alpha=0.8)

- Crea una instancia de la clase `Poly3DCollection`.

- `caras`: Se le pasa la lista de caras del cubo; cada cara es una lista de coordenadas de vértices, definiendo los polígonos a dibujar.

- `facecolors='lightcoral'`: Establece el color de relleno de las caras del cubo como 'lightcoral'.

- `edgecolors='black'`: Establece el color de las aristas (bordes) de las caras como 'black'.

- `alpha=0.8`: Establece la opacidad de las caras en 0.8 (80% opaco, 20% transparente).

- El objeto `Poly3DCollection` resultante, que representa visualmente el cubo, se asigna a la variable `poly`.

Línea 60: ax.add_collection3d(poly)

- Llama al método `add_collection3d` del objeto de ejes 3D `ax`.

- Se le pasa el objeto `poly` (la colección de polígonos del cubo). Esto añade el cubo a los ejes `ax` para que sea renderizado en el gráfico.

Línea 61: (En blanco)

- Línea en blanco: Separa la adición del cubo de la configuración de los límites y etiquetas de los ejes.

Línea 62: # Establecer límites y etiquetas de los ejes

- Comentario que indica que las siguientes líneas configurarán la apariencia de los ejes X, Y y Z del gráfico.

Línea 63: ax.set_xlim(-3, 3)

- Llama al método `set_xlim` del objeto `ax` para establecer los límites visibles del eje X entre -3 y 3.

Línea 64: ax.set_ylim(-3, 3)

- Llama al método `set_ylim` del objeto `ax` para establecer los límites visibles del eje Y entre -3 y 3.

Línea 65: ax.set_zlim(-3, 3)

- Llama al método `set_zlim` del objeto `ax` para establecer los límites visibles del eje Z entre -3 y 3. Estos límites aseguran que el cubo (cuyos vértices van de -1 a 1, pero pueden sesgarse un poco) sea completamente visible.

Línea 66: ax.set_xlabel('X')

- Llama al método `set_xlabel` del objeto `ax` para establecer la etiqueta del eje X como la cadena 'X'.

Línea 67: ax.set_ylabel('Y')

- Llama al método `set_ylabel` del objeto `ax` para establecer la etiqueta del eje Y como la cadena 'Y'.

Línea 68: ax.set_zlabel('Z')

- Llama al método `set_zlabel` del objeto `ax` para establecer la etiqueta del eje Z como la cadena 'Z'.

Línea 69: (En blanco)

- Línea en blanco: Separa la configuración de los ejes de la definición de la función `update` para la animación.

Línea 70: def update(frame):

- Define la función `update`, que será el corazón de la animación. Acepta un parámetro `frame`. `FuncAnimation` llamará a esta función repetidamente, pasando valores sucesivos para `frame` (según se configure más adelante).

Línea 71: """

- Inicio del docstring de la función `update`.

Línea 72: Actualiza el sesgado del cubo y la vista del gráfico.

- Línea del docstring: Describe el propósito general de la función `update`: modificar la forma del cubo (aplicando sesgado) y ajustar la perspectiva de la cámara en cada fotograma.

Línea 73: - Se aplica un factor de sesgado que oscila de forma sinusoidal.

- Línea del docstring: Detalla que el `shear_factor` se calculará usando una función seno para crear una oscilación suave.

Línea 74:  - Se aplica la transformación de sesgado a los vértices del cubo.

- Línea del docstring: Detalla que se usarán los vértices originales y el factor de sesgado calculado para obtener los vértices transformados.

Línea 75:  - Se actualizan las caras del cubo y se rota la vista (ángulo azimutal).

- Línea del docstring: Detalla que la geometría del objeto `Poly3DCollection` se actualizará con los nuevos vértices y que la cámara rotará horizontalmente.

Línea 76:  """

- Fin del docstring de la función `update`.

Línea 77:  # Oscilación del factor de sesgado

- Dentro de la función `update`: Comentario que indica que la siguiente línea calcula el valor del factor de sesgado para el fotograma actual.

Línea 78:  shear_factor = 0.5 * np.sin(np.deg2rad(frame))

- Dentro de la función `update`: Calcula el `shear_factor`.

- `np.deg2rad(frame)`: Convierte el valor de `frame` (que se espera esté en grados) a radianes, ya que las funciones trigonométricas de NumPy trabajan con radianes.

- `np.sin(...)`: Calcula el seno del ángulo en radianes. El resultado varía entre -1 y 1.

- `0.5 * ...`: Multiplica el resultado del seno por 0.5, haciendo que `shear_factor` oscile suavemente entre -0.5 y +0.5 a medida que `frame` cambia.

- El valor calculado se asigna a la variable `shear_factor`.

Línea 79:  vertices_sesgados = sesgar_x_z(vertices, shear_factor)

- Dentro de la función `update`: Llama a la función `sesgar_x_z`.

- Le pasa los `vertices` *originales* del cubo (que fueron definidos fuera de `update` y no cambian) y el `shear_factor` recién calculado para este fotograma.

- El array NumPy con las coordenadas de los vértices transformados (sesgados) que devuelve `sesgar_x_z` se asigna a la variable `vertices_sesgados`.

Línea 80: (En blanco)

- Dentro de la función `update`: Línea en blanco para separar el cálculo de los vértices sesgados de la actualización de la estructura de las caras.

Línea 81:  # Recalcular las caras con los nuevos vértices transformados

- Dentro de la función `update`: Comentario que indica que es necesario reconstruir la estructura de datos de las caras utilizando las coordenadas actualizadas de los vértices. `Poly3DCollection` espera una lista de listas de vértices.

Línea 82:  caras_sesgadas = [[vertices_sesgados[j] for j in [0, 1, 2, 3]],

- Dentro de la función `update`: Comienza la definición de la variable `caras_sesgadas`, que será una lista similar a la variable `caras` original.

- Se usa una lista por comprensión para crear la primera cara (trasera), pero esta vez extrayendo los vértices del array `vertices_sesgados` (los transformados).

Línea 83:  [vertices_sesgados[j] for j in [4, 5, 6, 7]],

- Define la segunda cara (frontal) usando los `vertices_sesgados` correspondientes.

Línea 84:  [vertices_sesgados[j] for j in [0, 1, 5, 4]],

- Define la tercera cara (inferior) usando los `vertices_sesgados` correspondientes.

Línea 85:  [vertices_sesgados[j] for j in [2, 3, 7, 6]],

- Define la cuarta cara (superior) usando los `vertices_sesgados` correspondientes.

Línea 86: [vertices_sesgados[j] for j in [1, 2, 6, 5]],

- Define la quinta cara (derecha) usando los `vertices_sesgados` correspondientes.

Línea 87: [vertices_sesgados[j] for j in [4, 7, 3, 0]]]

- Define la sexta y última cara (izquierda) usando los `vertices_sesgados` correspondientes.

- El `]]` cierra la definición de la lista `caras_sesgadas`.

Línea 88: poly.set_verts(caras_sesgadas)

- Dentro de la función `update`: Llama al método `set_verts` del objeto `Poly3DCollection` `poly` (el que representa el cubo en el gráfico).

- Se le pasa `caras_sesgadas`, que es la lista de caras definidas con las coordenadas de los vértices actualizadas para este fotograma. Este método modifica eficientemente la geometría del objeto `poly` existente para reflejar la transformación.

Línea 89: (En blanco)

- Dentro de la función `update`: Línea en blanco para separar la actualización de la geometría del cubo de la actualización de la vista y el título.

Línea 90: # Actualizar la vista: rotación de la cámara para un efecto dinámico

- Dentro de la función `update`: Comentario que explica que la siguiente línea ajustará el punto de vista desde el cual se observa el cubo.

Línea 91: ax.view_init(elev=30, azim=frame)

- Dentro de la función `update`: Llama al método `view_init` del objeto de ejes 3D `ax`.

- `elev=30`: Fija el ángulo de elevación de la cámara (la inclinación vertical) a 30 grados.

- `azim=frame`: Establece el ángulo azimutal (la rotación horizontal alrededor del eje Z) igual al valor actual de `frame`. Como `frame` varía (de 0 a 358 en este caso), esto hace que la cámara gire alrededor del cubo a medida que la animación progresa.

Línea 92: ax.set_title(f"Sesgado: factor {shear_factor:.2f}")

- Dentro de la función `update`: Llama al método `set_title` del objeto `ax` para actualizar el título del gráfico en cada fotograma.

- Utiliza una f-string (cadena formateada) para crear el título. Incluye el texto "Sesgado: factor " seguido del valor actual de `shear_factor`, formateado para mostrar solo dos decimales (`:.2f`).

Línea 93: return poly,

- Dentro de la función `update`: La sentencia `return`. Devuelve el objeto `poly` (el `Poly3DCollection` que fue modificado).

- La coma `,` al final es importante: asegura que se devuelve una tupla `(poly,)` en lugar de solo el objeto `poly`. `FuncAnimation` espera una secuencia iterable (como una tupla o lista) de los elementos gráficos ("artistas") que han sido modificados en la función `update`, especialmente si se usa `blit=True` (aunque aquí es `False`, devolverlo así es una práctica común).

Línea 94: (En blanco)

- Línea en blanco: Separa la definición de la función `update` de la línea que crea el objeto de animación.

Línea 95: # Crear la animación: el parámetro 'frame' varía de 0 a 360º en pasos de 2º

- Comentario que explica la finalidad de la siguiente línea: instanciar `FuncAnimation` para generar la animación. También aclara cómo se generarán los valores para el parámetro `frame` que recibirá la función `update`.

Línea 96: anim = FuncAnimation(fig, update, frames=np.arange(0, 360, 2), interval=50, blit=False)

- Crea la animación usando la clase `FuncAnimation` y asigna el objeto resultante a la variable `anim`.

- `fig`: El objeto `Figure` de Matplotlib sobre el cual se realizará la animación.

- `update`: La función (`update`) que se llamará para generar cada fotograma.

- `frames=np.arange(0, 360, 2)`: Define la secuencia de valores que se pasarán uno por uno como argumento `frame` a la función `update`. `np.arange(0, 360, 2)` genera un array NumPy con los números 0, 2, 4, ..., 358. Estos valores se usarán para calcular el `shear_factor` y el ángulo `azim`.

- `interval=50`: Especifica el retraso entre fotogramas en milisegundos. 50 ms equivale a 20 fotogramas por segundo (1000 ms / 50 ms/fotograma).

- `blit=False`: Controla si se usa la optimización de "blitting". `blit=True` intenta redibujar solo las partes del gráfico que han cambiado, lo que puede ser más rápido pero a veces causa problemas (especialmente con gráficos 3D o cambios en ejes/títulos). `blit=False` asegura que todo el gráfico se redibuje en cada fotograma, lo que es más robusto aunque potencialmente más lento.

Línea 97: (En blanco)

- Línea en blanco: Separa la creación del objeto `Animation` de la llamada final para mostrar el gráfico.

Línea 98: # Mostrar la animación

- Comentario que indica que la siguiente línea ejecutará la animación y mostrará la ventana del gráfico.

Línea 99: plt.show()

- Llama a la función `show()` del módulo `pyplot` (`plt`). Esta función inicia el bucle de eventos de Matplotlib, abre la ventana de la figura (`fig`) y comienza a mostrar la

animación (`anim`) ejecutando la función `update` para cada valor en `frames` con el intervalo especificado. La ejecución del script se pausará en esta línea hasta que el usuario cierre la ventana del gráfico.

## Resumen del Código

Este código Python utiliza las bibliotecas numpy y matplotlib para crear y mostrar una animación en 3D de un cubo que se deforma (sesga) continuamente mientras la cámara rota a su alrededor.

1. **Definición del Cubo:** Primero, define una función (crear_cubo) que genera las coordenadas de los 8 vértices de un cubo centrado en el origen y una estructura de datos que describe las 6 caras del cubo conectando estos vértices.
2. **Transformación de Sesgado:** Luego, define una función (sesgar_x_z) que implementa una transformación de sesgado. Esta transformación modifica la coordenada X de cada vértice basándose en su coordenada Z y un factor de sesgado (shear_factor), creando un efecto de inclinación.
3. **Configuración de la Visualización:** Se crea una figura de Matplotlib y un eje de trazado 3D. Se llama a crear_cubo para obtener la geometría inicial del cubo y se utiliza Poly3DCollection para crear un objeto gráfico que representa el cubo, añadiéndolo al eje 3D. Se configuran los límites y etiquetas de los ejes.
4. **Función de Actualización (Animación):** Se define una función update que se ejecutará para cada fotograma de la animación. En cada llamada:
    - Calcula un shear_factor que varía sinusoidalmente con el número de fotograma, creando una oscilación suave en la deformación.
    - Aplica este shear_factor a los vértices *originales* del cubo usando la función sesgar_x_z.
    - Actualiza la geometría del objeto Poly3DCollection con los vértices sesgados.

- Gira la vista de la cámara (ángulo azimutal) basándose en el número de fotograma.
- Actualiza el título del gráfico para mostrar el factor de sesgado actual.

5. **Creación y Ejecución de la Animación:** Se utiliza FuncAnimation para crear el objeto de animación. Se le indica que use la figura creada, llame a la función update repetidamente, genere números de fotograma de 0 a 358 (en pasos de 2), espere 50 milisegundos entre fotogramas y redibuje toda la figura en cada paso (blit=False).
6. **Visualización:** Finalmente, plt.show() abre la ventana de Matplotlib y muestra la animación resultante: un cubo que se inclina hacia adelante y hacia atrás mientras la vista gira a su alrededor.