

1. **Línea 1:** `import numpy as np`
Importa la librería NumPy y la renombra como np, facilitando operaciones matemáticas y con arrays.
2. **Línea 2:** `import matplotlib`
Importa la librería Matplotlib, que se utiliza para crear gráficos.
3. **Línea 3:** `matplotlib.use("TkAgg")`
Configura Matplotlib para usar el backend "TkAgg", permitiendo la visualización interactiva en ventanas.
4. **Línea 4:** `import matplotlib.pyplot as plt`
Importa el módulo pyplot de Matplotlib, asignándole el alias plt, para facilitar la creación de gráficos.
5. **Línea 5:** `from mpl_toolkits.mplot3d.art3d import Poly3DCollection`
Importa la clase Poly3DCollection para representar colecciones de polígonos en un espacio 3D.
6. **Línea 6:** `from matplotlib.animation import FuncAnimation`
Importa FuncAnimation, que permite crear animaciones actualizando la figura a intervalos regulares.
7. **Línea 7:** (línea en blanco)
Línea en blanco para separar bloques de código.
8. **Línea 8:** `def crear_cubo():`
Define la función crear_cubo() que generará los vértices y las caras de un cubo.
9. **Línea 9:** `"""`
Inicio del docstring que describe la función crear_cubo.
10. **Línea 10:** Crea un cubo centrado en el origen con lados de longitud 2.
Explica que la función crea un cubo centrado en el origen con lados de longitud 2.
11. **Línea 11:** Devuelve:
Indica que a continuación se describen los valores que devuelve la función.
12. **Línea 12:** `vertices: array con las coordenadas (x, y, z) de cada vértice.`
Describe que se devolverá un array con las coordenadas de cada vértice del cubo.

13. **Línea 13:** caras: lista de caras del cubo, donde cada cara es una lista de vértices.
Describe que se devolverá una lista de caras, cada una compuesta por los vértices correspondientes.
14. **Línea 14:** ""
Fin del docstring de la función crear_cubo.
15. **Línea 15:** # Definir los vértices del cubo
Comentario que indica que a continuación se definen los vértices del cubo.
16. **Línea 16:** vertices = np.array([[-1, -1, -1],
Crea un array de NumPy y define el primer vértice del cubo en (-1, -1, -1).
17. **Línea 17:** [1, -1, -1],
Añade el segundo vértice en (1, -1, -1).
18. **Línea 18:** [1, 1, -1],
Añade el tercer vértice en (1, 1, -1).
19. **Línea 19:** [-1, 1, -1],
Añade el cuarto vértice en (-1, 1, -1).
20. **Línea 20:** [-1, -1, 1],
Añade el quinto vértice en (-1, -1, 1).
21. **Línea 21:** [1, -1, 1],
Añade el sexto vértice en (1, -1, 1).
22. **Línea 22:** [1, 1, 1],
Añade el séptimo vértice en (1, 1, 1).
23. **Línea 23:** [-1, 1, 1]])
Añade el octavo vértice en (-1, 1, 1) y cierra la definición del array.
24. **Línea 24:** (línea en blanco)
Línea en blanco para separar secciones dentro de la función.
25. **Línea 25:** # Definir las caras del cubo usando los índices de los vértices
Comentario que indica que se definirán las caras del cubo utilizando los índices de los vértices.
26. **Línea 26:** caras = [[vertices[j] for j in [0, 1, 2, 3]], # Cara trasera
Define la primera cara (trasera) del cubo usando los vértices de los índices 0, 1, 2 y 3; se utiliza una lista por comprensión y se agrega un comentario.

27. **Línea 27:** [vertices[j] for j in [4, 5, 6, 7]], # Cara frontal
Define la segunda cara (frontal) usando los vértices en índices 4, 5, 6 y 7, con comentario.
28. **Línea 28:** [vertices[j] for j in [0, 1, 5, 4]], # Cara inferior
Define la tercera cara (inferior) utilizando los vértices en índices 0, 1, 5 y 4.
29. **Línea 29:** [vertices[j] for j in [2, 3, 7, 6]], # Cara superior
Define la cuarta cara (superior) usando los vértices en índices 2, 3, 7 y 6.
30. **Línea 30:** [vertices[j] for j in [1, 2, 6, 5]], # Cara derecha
Define la quinta cara (derecha) utilizando los vértices en índices 1, 2, 6 y 5.
31. **Línea 31:** [vertices[j] for j in [4, 7, 3, 0]] # Cara izquierda]
Define la sexta cara (izquierda) usando los vértices en índices 4, 7, 3 y 0 y cierra la lista de caras, con comentario.
32. **Línea 32:** return vertices, caras
La función retorna el array de vértices y la lista de caras creados.
33. **Línea 33:** (línea en blanco)
Línea en blanco para separar definiciones de funciones.
34. **Línea 34:** def rotacion_y(vertices, angulo):
Define la función rotacion_y() que aplicará una rotación alrededor del eje Y a los vértices.
35. **Línea 35:** ""
Inicio del docstring que describe la función rotacion_y.
36. **Línea 36:** Aplica una rotación al cubo alrededor del eje Y.
Explica brevemente la funcionalidad de la función: rotar el cubo en torno al eje Y.
37. **Línea 37:**
Línea en blanco dentro del docstring para separar secciones.
38. **Línea 38:** Parámetros:
Indica el inicio de la descripción de los parámetros de la función.
39. **Línea 39:** vertices: array de vértices (n, 3).
Describe el parámetro vertices como un array de forma (n, 3) que contiene las coordenadas de cada vértice.

40. **Línea 40:** `angulo`: ángulo de rotación en radianes.
Describe el parámetro `angulo`, que es el valor en radianes para la rotación.
41. **Línea 41:** `Retorna`:
Indica el inicio de la descripción del valor de retorno de la función.
42. **Línea 42:** `vertices_rotados`: array con los vértices rotados.
Explica que la función retorna un array de vértices tras aplicar la rotación.
43. **Línea 43:** `"""`
Fin del docstring de la función `rotacion_y`.
44. **Línea 44:** `# Matriz de rotación para el eje Y`
Comentario que indica que se definirá la matriz de rotación para el eje Y.
45. **Línea 45:** `cos_a = np.cos(angulo)`
Calcula el coseno del ángulo y lo asigna a la variable `cos_a`.
46. **Línea 46:** `sin_a = np.sin(angulo)`
Calcula el seno del ángulo y lo asigna a la variable `sin_a`.
47. **Línea 47:** `R_y = np.array([[cos_a, 0, sin_a],`
Define la primera fila de la matriz de rotación para el eje Y usando `cos_a` y `sin_a`.
48. **Línea 48:** `[0, 1, 0],`
Define la segunda fila de la matriz, que deja invariante la coordenada Y.
49. **Línea 49:** `[-sin_a, 0, cos_a]])`
Define la tercera fila de la matriz de rotación, completando la transformación en el eje Y.
50. **Línea 50:** `# Se aplica la rotación a cada vértice`
Comentario que indica que se procederá a aplicar la rotación a los vértices usando la matriz definida.
51. **Línea 51:** `return vertices.dot(R_y.T)`
Retorna el resultado de multiplicar el array de vértices por la transpuesta de `R_y`, aplicando la rotación a cada vértice.
52. **Línea 52:** (línea en blanco)
Línea en blanco para separar secciones de código.

53. **Línea 53:** # Configuración de la figura y el eje 3D
Comentario que indica que a continuación se configurará la figura y el eje para la visualización 3D.
54. **Línea 54:** fig = plt.figure()
Crea una nueva figura para la gráfica y la asigna a la variable fig.
55. **Línea 55:** ax = fig.add_subplot(111, projection="3d")
Agrega un subplot 3D a la figura, asignándolo a la variable ax.
56. **Línea 56:** (línea en blanco)
Línea en blanco para separar secciones.
57. **Línea 57:** # Crear el cubo inicial
Comentario que indica que se va a crear el cubo inicial.
58. **Línea 58:** vertices, caras = crear_cubo()
Llama a la función crear_cubo() y asigna los resultados a las variables vertices y caras.
59. **Línea 59:** poly = Poly3DCollection(caras, facecolors="lightblue",
edgecolors="black", alpha=0.8)
Crea un objeto Poly3DCollection con las caras del cubo, asignándole color de cara azul claro, bordes negros y una opacidad del 80%.
60. **Línea 60:** ax.add_collection3d(poly)
Agrega el objeto 3D poly al eje ax para que se visualice en el gráfico.
61. **Línea 61:** (línea en blanco)
Línea en blanco para separar secciones.
62. **Línea 62:** # Establecer límites y etiquetas del gráfico
Comentario que indica que se configurarán los límites de los ejes y sus etiquetas.
63. **Línea 63:** ax.set_xlim(-3, 3)
Configura el límite del eje X entre -3 y 3.
64. **Línea 64:** ax.set_ylim(-3, 3)
Configura el límite del eje Y entre -3 y 3.
65. **Línea 65:** ax.set_zlim(-3, 3)
Configura el límite del eje Z entre -3 y 3.

66. **Línea 66:** `ax.set_xlabel("X")`
Establece la etiqueta del eje X como "X".
67. **Línea 67:** `ax.set_ylabel("Y")`
Establece la etiqueta del eje Y como "Y".
68. **Línea 68:** `ax.set_zlabel("Z")`
Establece la etiqueta del eje Z como "Z".
69. **Línea 69:** (línea en blanco)
Línea en blanco para separar secciones.
70. **Línea 70:** `def update(frame):`
Define la función `update()` que se encargará de actualizar el estado del cubo en cada cuadro de la animación.
71. **Línea 71:** `"""`
Inicio del docstring para la función `update`.
72. **Línea 72:** Actualiza la rotación del cubo y la vista del gráfico.
Describe que la función actualiza la rotación del cubo y la vista de la cámara.
73. **Línea 73:** - Se rota el cubo alrededor del eje Y.
Indica que la función rota el cubo alrededor del eje Y.
74. **Línea 74:** - Se actualiza el ángulo de la cámara para obtener un efecto dinámico.
Indica que también se actualiza el ángulo de la cámara para mejorar la visualización.
75. **Línea 75:** `"""`
Fin del docstring de la función `update`.
76. **Línea 76:** `# Convertir el ángulo a radianes`
Comentario que explica que se convertirá el ángulo del frame a radianes.
77. **Línea 77:** `angulo = np.deg2rad(-frame * 3)`
Convierte a radianes el valor calculado a partir del frame (multiplicado por -3) y lo asigna a `angulo`.
78. **Línea 78:** `# Rotar los vértices del cubo`
Comentario que indica que se procederá a rotar los vértices del cubo.

79. **Línea 79:** `vertices_rot = rotacion_y(vertices, angulo)`
Llama a la función `rotacion_y()` con los vértices originales y el ángulo para obtener los vértices rotados.
80. **Línea 80:** (línea en blanco)
Línea en blanco para separar secciones dentro de la función `update`.
81. **Línea 81:** `# Recalcular las caras con los nuevos vertices rotados`
Comentario que indica que se recalcularán las caras del cubo usando los nuevos vértices rotados.
82. **Línea 82:** `caras_rotadas = [[vertices_rot[j] for j in [0, 1, 2, 3]],`
Define la primera cara (trasera) del cubo con los vértices rotados.
83. **Línea 83:** `[vertices_rot[j] for j in [4, 5, 6, 7]],`
Define la segunda cara (frontal) usando los vértices rotados.
84. **Línea 84:** `[vertices_rot[j] for j in [0, 1, 5, 4]],`
Define la tercera cara (inferior) usando los vértices rotados.
85. **Línea 85:** `[vertices_rot[j] for j in [2, 3, 7, 6]],`
Define la cuarta cara (superior) usando los vértices rotados.
86. **Línea 86:** `[vertices_rot[j] for j in [1, 2, 6, 5]],`
Define la quinta cara (derecha) usando los vértices rotados.
87. **Línea 87:** `[vertices_rot[j] for j in [4, 7, 3, 0]]]`
Define la sexta cara (izquierda) usando los vértices rotados y cierra la lista de caras.
88. **Línea 88:** `poly.set_verts(caras_rotadas)`
Actualiza el objeto `poly` con las nuevas caras calculadas, para reflejar la rotación.
89. **Línea 89:** (línea en blanco)
Línea en blanco para separar secciones dentro de la función `update`.
90. **Línea 90:** `# Actualizar la vista: se rota la cámara para mejorar al visualización dinámica`
Comentario que indica que se actualizará la vista de la cámara para un efecto dinámico.

91. **Línea 91:** `ax.view_init(elev=30, azim=frame)`
Ajusta la vista de la cámara fijando la elevación en 30° y usando el valor del frame para el ángulo azimutal.
92. **Línea 92:** `ax.set_title(f"Rotación: {frame}°")`
Actualiza el título del gráfico para mostrar el ángulo de rotación actual.
93. **Línea 93:** `return (poly,)`
Retorna una tupla con el objeto poly, necesario para la actualización en la animación.
94. **Línea 94:** (línea en blanco)
Línea en blanco para separar secciones.
95. **Línea 95:** `# Crear la animación: el ángulo de 0 a 360 con incrementos de 2 grados`
Comentario que indica que se creará una animación con frames desde 0 hasta 360 grados en pasos de 2.
96. **Línea 96:** `anim = FuncAnimation(fig, update, frames=np.arange(0, 360, 2), interval=50, blit=False)`
Crea la animación utilizando FuncAnimation, pasando la figura, la función de actualización, los frames (de 0 a 360 con paso 2), un intervalo de 50 ms entre frames y deshabilitando el blit.
97. **Línea 97:** (línea en blanco)
Línea en blanco para separar secciones.
98. **Línea 98:** `# Mostrar la animación`
Comentario que indica que se mostrará la animación en una ventana.
99. **Línea 99:** `plt.show()`
Llama a `plt.show()` para mostrar la ventana con la animación del cubo 3D.
-

Explicación Resumida General

El código crea y anima un cubo 3D usando las librerías NumPy y Matplotlib. Inicialmente, se define una función para generar un cubo (sus vértices y caras) y otra para aplicar una rotación alrededor del eje Y. Se configura una figura 3D, se dibuja el cubo y se definen los límites y etiquetas de los ejes. La función `update` se encarga de rotar el cubo y actualizar la vista de la cámara en cada cuadro de la animación.

Finalmente, se crea una animación con FuncAnimation y se muestra la ventana de la animación.

