**Importación de módulos**

1. import numpy as np
   * Importa la biblioteca numpy y la asigna al alias np. Se usa para manejar arreglos numéricos y operaciones matemáticas.
2. import matplotlib.pyplot as plt
   * Importa pyplot de matplotlib con el alias plt, permitiendo crear gráficos y visualizaciones.
3. from mpl\_toolkits.mplot3d.art3d import Poly3DCollection
   * Importa Poly3DCollection del módulo mpl\_toolkits.mplot3d.art3d, que permite crear colecciones de polígonos en 3D.
4. from matplotlib.animation import FuncAnimation
   * Importa FuncAnimation de matplotlib.animation, que se usa para animar gráficos en matplotlib.

**Definición de la función generar\_piramide**

1. def generar\_piramide(base\_size=1, altura=1.5):
   * Define la función generar\_piramide con parámetros opcionales base\_size (tamaño de la base) y altura (altura de la pirámide).
2. base = np.array(
   * Declara una variable base y la define como un arreglo de numpy.
3. [
   * Abre la lista que contendrá los vértices de la base de la pirámide.
4. [-base\_size, -base\_size, 0],
   * Define el primer vértice de la base en la coordenada (-base\_size, -base\_size, 0).
5. [base\_size, -base\_size, 0],
   * Define el segundo vértice de la base en la coordenada (base\_size, -base\_size, 0).
6. [base\_size, base\_size, 0],
   * Define el tercer vértice de la base en la coordenada (base\_size, base\_size, 0).
7. [-base\_size, base\_size, 0],
   * Define el cuarto vértice de la base en la coordenada (-base\_size, base\_size, 0).
8. ]
   * Cierra la lista de vértices de la base.
9. )
   * Cierra la declaración del arreglo np.array.
10. # Punto superior de la pirámide con variación procedural
    * Comentario que indica que se generará el vértice superior de la pirámide con un pequeño ajuste aleatorio.
11. peak = np.array([0, 0, altura + np.random.uniform(-0.2, 0.2)])
    * Define peak, el punto superior de la pirámide, con coordenadas (0,0,altura) más una variación aleatoria entre -0.2 y 0.2.
12. # Definir las caras de la pirámide
    * Comentario indicando que se definirán las caras de la pirámide.
13. caras = [
    * Declara la lista caras, que contendrá los polígonos de la pirámide.
14. [base[0], base[1], base[2], base[3]],
    * Define la base de la pirámide como un cuadrado usando los cuatro vértices de base.
15. [base[0], base[1], peak],
    * Define la primera cara lateral, conectando los dos primeros vértices de la base con el peak.
16. [base[1], base[2], peak],
    * Define la segunda cara lateral, conectando los siguientes dos vértices con el peak.
17. [base[2], base[3], peak],
    * Define la tercera cara lateral con los siguientes dos vértices y el peak.
18. [base[3], base[0], peak],
    * Define la cuarta y última cara lateral conectando los últimos dos vértices con el peak.
19. ]
    * Cierra la lista caras.
20. return caras
    * Devuelve la lista de caras de la pirámide.

**Creación de la figura 3D**

1. # Crear la figura y el eje 3D
   * Comentario indicando que se creará la figura y el eje en 3D.
2. fig = plt.figure()
   * Crea una figura de matplotlib y la almacena en la variable fig.
3. ax = fig.add\_subplot(111, projection="3d")
   * Agrega un subplot a fig con proyección 3d y lo almacena en ax.

**Generación de múltiples pirámides**

1. # Generar y dibujar múltiples pirámides procedurales
   * Comentario indicando que se van a generar varias pirámides en una cuadrícula.
2. for i in range(5):
   * Inicia un bucle for que iterará 5 veces para colocar pirámides en el eje X.
3. for j in range(5):
   * Inicia un segundo bucle for anidado que iterará 5 veces para colocar pirámides en el eje Y.
4. offset\_x, offset\_y = i \* 3, j \* 3
   * Calcula el desplazamiento en x y y, separando las pirámides cada 3 unidades.
5. piramide = generar\_piramide()
   * Llama a la función generar\_piramide() para generar una pirámide y la almacena en piramide.
6. # Dibujar cada pirámide con un desplazamiento
   * Comentario indicando que se van a dibujar las pirámides con desplazamiento.
7. collection = Poly3DCollection(
   * Crea un objeto Poly3DCollection para representar la pirámide en 3D.
8. [
   * Abre la lista de caras para la colección de polígonos.
9. [np.array(p) + [offset\_x, offset\_y, 0] for p in cara]
   * Ajusta las coordenadas de cada punto de la pirámide según su offset\_x y offset\_y.
10. for cara in piramide
    * Itera sobre cada cara de la pirámide para aplicarle el desplazamiento.
11. ],
    * Cierra la lista de caras desplazadas.
12. facecolors="lightblue",
    * Asigna un color azul claro a las caras de la pirámide.
13. edgecolors="black",
    * Asigna un color negro a los bordes de la pirámide.
14. alpha=0.6,
    * Asigna transparencia 0.6 a las pirámides.
15. )
    * Cierra la definición de Poly3DCollection.
16. ax.add\_collection(collection)
    * Agrega la pirámide a la figura 3D.

**Configuración del gráfico**

1. ax.set\_xlim(-2, 15)
   * Define los límites del eje X de -2 a 15.
2. ax.set\_ylim(-2, 15)
   * Define los límites del eje Y de -2 a 15.
3. ax.set\_zlim(0, 2)
   * Define los límites del eje Z de 0 a 2.
4. ax.set\_xlabel("X")
   * Establece la etiqueta del eje X.
5. ax.set\_ylabel("Y")
   * Establece la etiqueta del eje Y.
6. ax.set\_zlabel("Z")
   * Establece la etiqueta del eje Z.
7. ax.set\_title("Modelado Procedural - Pirámides")
   * Establece el título del gráfico.

**Animación**

1. def update(frame):
   * Define la función update para actualizar la vista de la animación.
2. ax.view\_init(elev=30, azim=frame)
   * Cambia el ángulo azim (rotación en Z) en cada frame.
3. ani = FuncAnimation(fig, update, frames=np.arange(0, 360, 2), interval=10)
   * Crea la animación girando la vista 3D.

**Mostrar la figura**

1. plt.show()
   * Muestra la figura con las pirámides y la animación.