Línea 1: import numpy as np

- Importa la biblioteca `numpy` y le asigna el alias `np`. NumPy se utilizará para operaciones numéricas, especialmente para crear arrays que representen los vértices del hexágono, la malla de puntos y los colores del degradado.

Línea 2: import matplotlib.pyplot as plt

- Importa el submódulo `pyplot` de la biblioteca `matplotlib` y le asigna el alias `plt`. Pyplot se usará para crear la figura, los ejes y mostrar el gráfico final.

Línea 3: from matplotlib.path import Path

- Importa específicamente la clase `Path` del módulo `matplotlib.path`. La clase `Path` se usa para representar una forma geométrica (en este caso, el hexágono) y permite realizar operaciones como verificar si puntos están dentro de esa forma.

Línea 4: import matplotlib.patches as patches

- Importa el módulo `matplotlib.patches` y le asigna el alias `patches`. Este módulo contiene clases para dibujar formas geométricas ("patches") en un gráfico de Matplotlib, como polígonos. Se usará para dibujar el contorno del hexágono.

Línea 5: (En blanco)

- Línea en blanco: Separa el bloque de importaciones de la definición de la función.

Línea 6: def draw_hexagon_gradient():

- Define una función llamada `draw_hexagon_gradient` que no acepta ningún parámetro. Todo el proceso de dibujo se realizará dentro de esta función.

Línea 7: fig, ax = plt.subplots()

- Dentro de la función `draw_hexagon_gradient`: Llama a `plt.subplots()` para crear una nueva figura (`fig`) y un conjunto de ejes (`ax`) dentro de esa figura. `ax` es el objeto sobre el que se dibujarán los elementos gráficos.

Línea 8: ax.set_aspect("equal")

- Dentro de la función `draw_hexagon_gradient`: Establece la relación de aspecto de los ejes `ax` a "equal". Esto asegura que una unidad en el eje X tenga la misma longitud visual que una unidad en el eje Y, evitando que el hexágono se vea distorsionado (estirado o achatado).

Línea 9: ax.axis("off")

- Dentro de la función `draw_hexagon_gradient`: Oculta los ejes (las líneas X e Y, las marcas y las etiquetas) del gráfico, dejando solo el contenido dibujado.

Línea 10: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar la configuración inicial de los ejes del cálculo de los vértices.

Línea 11: # Definir los vértices del hexágono (inscrito en un círculo de radio 1)

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica que el siguiente bloque de código calculará las coordenadas de los vértices de un hexágono regular.

Línea 12: theta = np.linspace(0, 2 * np.pi, 7)

- Dentro de la función `draw_hexagon_gradient`: Usa `np.linspace` para crear un array NumPy llamado `theta`. Este array contiene 7 valores espaciados uniformemente entre 0 y 2π (un círculo completo). Estos representan los ángulos de los 6 vértices del hexágono más el punto inicial repetido al final para cerrar la forma si fuera necesario para algunos cálculos (aunque `np.column_stack` y `Path` manejan esto correctamente).

Línea 13: vertices = np.column_stack([np.cos(theta), np.sin(theta)])

- Dentro de la función `draw_hexagon_gradient`: Calcula las coordenadas (x, y) de los vértices.

  - `np.cos(theta)` calcula el coseno de cada ángulo en `theta` (coordenadas x en un círculo unitario).

  - `np.sin(theta)` calcula el seno de cada ángulo en `theta` (coordenadas y en un círculo unitario).

  - `np.column_stack([...])` toma las coordenadas x e y y las apila como columnas, creando un array 2D `vertices` donde cada fila es un par (x, y) de un vértice del hexágono (y el punto final duplicado). El resultado tiene forma (7, 2).

Línea 14: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar el cálculo de vértices de la creación de la malla.

Línea 15: # Crear una malla de puntos que cubre el área del hexágono

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica que el siguiente código creará una cuadrícula (malla) de puntos en el plano XY. El degradado se calculará para cada punto de esta malla.

Línea 16: nx, ny = 500, 500

- Dentro de la función `draw_hexagon_gradient`: Define las variables `nx` y `ny` con el valor 500. Estas representan la resolución de la malla en las direcciones X e Y (número de puntos).

Línea 17: x = np.linspace(-1, 1, nx)

- Dentro de la función `draw_hexagon_gradient`: Crea un array NumPy `x` con `nx` (500) puntos espaciados uniformemente entre -1 y 1. Estas serán las coordenadas X de la malla.

Línea 18: y = np.linspace(-1, 1, ny)

- Dentro de la función `draw_hexagon_gradient`: Crea un array NumPy `y` con `ny` (500) puntos espaciados uniformemente entre -1 y 1. Estas serán las coordenadas Y de la malla.

Línea 19: X, Y = np.meshgrid(x, y)

- Dentro de la función `draw_hexagon_gradient`: Usa `np.meshgrid` para crear dos arrays 2D, `X` e `Y`. `X` contiene las coordenadas x repetidas para cada fila, e `Y` contiene las coordenadas y repetidas para cada columna. Juntos, `X` e `Y` definen las coordenadas (x, y) de cada punto en la malla de 500x500.

Línea 20: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar la creación de la malla del cálculo del degradado.

Línea 21: # Crear el degradado: se varía el color según la coordenada x

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica la lógica del degradado: el color cambiará horizontalmente basándose en la posición x.

Línea 22: # Normalizar x para que varíe de 0 (izquierda) a 1 (derecha)

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica el paso de normalización necesario para mapear las coordenadas x (que van de -1 a 1) al rango 0-1, útil para definir los componentes de color.

Línea 23: norm_x = (X + 1) / 2

- Dentro de la función `draw_hexagon_gradient`: Calcula `norm_x`, un array 2D del mismo tamaño que `X` (500x500). Para cada punto en la malla `X`, suma 1 (el rango se vuelve 0 a 2) y divide por 2 (el rango se vuelve 0 a 1). `norm_x` ahora representa la posición horizontal normalizada de cada punto de la malla.

Línea 24: grad = np.empty((ny, nx, 3))

- Dentro de la función `draw_hexagon_gradient`: Crea un array NumPy vacío llamado `grad` con dimensiones (ny, nx, 3) o (500, 500, 3). Este array almacenará los valores de color RGB para cada punto de la malla (3 valores: Rojo, Verde, Azul). Se usa `np.empty` porque se llenarán todos los valores inmediatamente después.

Línea 25: grad[:, :, 0] = norm_x # Canal rojo aumenta de izquierda a derecha

- Dentro de la función `draw_hexagon_gradient`: Asigna los valores de `norm_x` al primer canal (índice 0, Rojo) del array `grad`. Como `norm_x` va de 0 a 1 de izquierda a derecha, el componente rojo del color aumentará en esa dirección.

Línea 26: grad[:, :, 1] = 0 # Canal verde en 0

- Dentro de la función `draw_hexagon_gradient`: Asigna el valor 0 a todos los elementos del segundo canal (índice 1, Verde) del array `grad`. El color no tendrá componente verde.

Línea 27: grad[:, :, 2] = 1 - norm_x # Canal azul disminuye de izquierda a derecha

- Dentro de la función `draw_hexagon_gradient`: Asigna los valores de `1 - norm_x` al tercer canal (índice 2, Azul) del array `grad`. Como `norm_x` va de 0 a 1 de izquierda a derecha, `1 - norm_x` va de 1 a 0. El componente azul será máximo a la izquierda y mínimo a la derecha. El resultado es un degradado de azul (izquierda) a rojo (derecha).

Línea 28: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar el cálculo del degradado de la creación de la máscara.

Línea 29: # Crear una máscara para que el degradado se aplique solo dentro del hexágono

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica que el siguiente código determinará qué puntos de la malla caen dentro de los límites del hexágono.

Línea 30: hex_path = Path(vertices)

- Dentro de la función `draw_hexagon_gradient`: Crea un objeto `Path` llamado `hex_path` a partir del array `vertices` que define el contorno del hexágono.

Línea 31: points = np.column_stack((X.ravel(), Y.ravel()))

- Dentro de la función `draw_hexagon_gradient`: Prepara los puntos de la malla para la comprobación con `Path`.

  - `X.ravel()` aplana el array 2D `X` en un array 1D.

  - `Y.ravel()` aplana el array 2D `Y` en un array 1D.

  - `np.column_stack(...)` combina estos dos arrays 1D en un array 2D `points` de forma (nx*ny, 2), donde cada fila es un par (x, y) de un punto de la malla.

Línea 32: mask = hex_path.contains_points(points).reshape((ny, nx))

- Dentro de la función `draw_hexagon_gradient`: Crea la máscara booleana.

  - `hex_path.contains_points(points)` llama al método `contains_points` del objeto `Path`. Devuelve un array 1D booleano, donde cada elemento es `True` si el punto correspondiente en `points` está dentro o en el borde del `hex_path`, y `False` si está fuera.

  - `.reshape((ny, nx))` cambia la forma de este array 1D booleano de vuelta a las dimensiones de la malla original (500x500). El array `mask` ahora tiene `True` en las posiciones correspondientes a puntos dentro del hexágono y `False` fuera.

Línea 33: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar la creación de la máscara de su aplicación.

Línea 34: # Establecer el degradado fuera del hexágono (por ejemplo, en blanco)

- Dentro de la función `draw_hexagon_gradient`: Comentario que explica que los puntos fuera del hexágono se colorearán de blanco.

Línea 35: grad[~mask] = [1, 1, 1]

- Dentro de la función `draw_hexagon_gradient`: Aplica la máscara al array de degradado `grad`.

  - `~mask`: Invierte la máscara booleana. Ahora `True` corresponde a puntos *fuera* del hexágono.

  - `grad[~mask]` selecciona todos los píxeles (y sus 3 componentes RGB) en el array `grad` donde `~mask` es `True`.

  - `= [1, 1, 1]`: Asigna el color blanco (R=1, G=1, B=1) a todos los píxeles seleccionados (los que están fuera del hexágono).

Línea 36: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar la aplicación de la máscara de la visualización de la imagen.

Línea 37: # Mostrar el degradado usando imshow

- Dentro de la función `draw_hexagon_gradient`: Comentario que indica que el array de colores `grad` se mostrará como una imagen.

Línea 38: ax.imshow(grad, extent=[-1, 1, -1, 1], origin="lower")

- Dentro de la función `draw_hexagon_gradient`: Usa el método `imshow` del objeto `ax` para mostrar el array 2D de colores `grad` como una imagen.

  - `grad`: El array (500, 500, 3) con los datos de color RGB.

  - `extent=[-1, 1, -1, 1]`: Define los límites de coordenadas (xmin, xmax, ymin, ymax) que corresponden a los bordes de la imagen `grad`. Esto asegura que la imagen se

alinee correctamente con las coordenadas usadas para definir el hexágono y la malla (-1 a 1 en ambos ejes).

- `origin="lower"`: Especifica que el origen del array `grad` (índice [0, 0]) corresponde a la esquina inferior izquierda del gráfico. Es necesario porque por defecto `imshow` asume el origen en la esquina superior izquierda.

Línea 39: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco para separar la visualización de la imagen del dibujo del contorno.

Línea 40: # Dibujar el contorno del hexágono

- Dentro de la función `draw_hexagon_gradient`: Comentario que indica que se dibujará el borde del hexágono sobre la imagen del degradado.

Línea 41: hexagon = patches.Polygon(

- Dentro de la función `draw_hexagon_gradient`: Comienza la creación de un objeto `Polygon` del módulo `patches`.

Línea 42: vertices, closed=True, fill=False, edgecolor="black", linewidth=2

- Dentro de la función `draw_hexagon_gradient`: Define las propiedades del polígono:

- `vertices`: El array con las coordenadas de los vértices del hexágono.

- `closed=True`: Indica que el último vértice debe conectarse con el primero para cerrar la forma.

- `fill=False`: Indica que el interior del polígono no debe rellenarse con color.

- `edgecolor="black"`: Establece el color del borde (contorno) a negro.

- `linewidth=2`: Establece el grosor de la línea del borde a 2 puntos.

Línea 43:)

- Dentro de la función `draw_hexagon_gradient`: Cierra la llamada al constructor `patches.Polygon`. El objeto polígono se asigna a la variable `hexagon`.

Línea 44: ax.add_patch(hexagon)

- Dentro de la función `draw_hexagon_gradient`: Llama al método `add_patch` del objeto `ax` para añadir el objeto `hexagon` (el contorno) al gráfico.

Línea 45: (En blanco)

- Dentro de la función `draw_hexagon_gradient`: Línea en blanco antes de mostrar el gráfico.

Línea 46: plt.show()

- Dentro de la función `draw_hexagon_gradient`: Llama a `plt.show()` para abrir la ventana de Matplotlib y mostrar el gráfico completo con el hexágono con degradado y su contorno.

Línea 47: (En blanco)

- Línea en blanco: Separa la definición de la función de la llamada a la misma.

Línea 48: (En blanco)

- Línea en blanco: Espacio adicional antes de la llamada a la función.

Línea 49: draw_hexagon_gradient()

- Llama a la función `draw_hexagon_gradient` definida anteriormente, ejecutando todo el código contenido en ella y produciendo el gráfico.

Resumen del Código

Este código utiliza numpy y matplotlib para dibujar un hexágono regular centrado en el origen. El interior del hexágono está relleno con un degradado de color que varía horizontalmente: es azul en el lado izquierdo, pasa por el magenta en el centro y llega a rojo en el lado derecho. El área fuera del hexágono se deja en blanco. Finalmente, se dibuja un contorno negro alrededor del hexágono y se muestra la imagen resultante.