Línea 1: import numpy as np

- Importa la biblioteca `numpy` y le asigna el alias `np`. Se utilizará para trabajar con arrays numéricos, especialmente para representar la imagen como una matriz de píxeles y para manejar los colores como arrays.

Línea 2: from PIL import Image, ImageDraw

- Importa las clases `Image` y `ImageDraw` desde la biblioteca `PIL` (Pillow, el fork moderno de Python Imaging Library). `Image` se usará para crear y manipular imágenes, y `ImageDraw` para dibujar formas (como el contorno del hexágono) sobre la imagen.

Línea 3: import matplotlib.pyplot as plt

- Importa el submódulo `pyplot` de la biblioteca `matplotlib` y le asigna el alias `plt`. Se usará para visualizar la imagen resultante después del proceso de relleno.

Línea 4: import math

- Importa el módulo `math`, que proporciona funciones matemáticas básicas, como `sin` y `cos`, necesarias para calcular los vértices del hexágono.

Línea 5: (En blanco)

- Línea en blanco: Separa el bloque de importaciones de la definición de la función `flood\_fill`.

Línea 6: def flood\_fill(image, x, y, new\_color):

- Define una función llamada `flood\_fill` que acepta cuatro parámetros: `image` (la imagen a modificar), `x` (la coordenada x del píxel inicial), `y` (la coordenada y del píxel inicial) y `new\_color` (el color con el que se rellenará).

Línea 7: """

- Inicio del docstring (cadena de documentación) de la función `flood\_fill`.

Línea 8: Realiza flood fill en 'image' a partir del píxel (x, y).

- Línea del docstring: Describe brevemente el propósito de la función: aplicar el algoritmo de relleno "flood fill".

Línea 9: image: numpy array de la imagen en formato RGB.

- Línea del docstring: Describe el parámetro `image`, especificando que debe ser un array NumPy que representa una imagen RGB.

Línea 10: new\_color: np.array con el nuevo color [R, G, B].

- Línea del docstring: Describe el parámetro `new\_color`, indicando que debe ser un array NumPy con los componentes [Rojo, Verde, Azul] del color de relleno.

Línea 11: """

- Fin del docstring de la función `flood\_fill`.

Línea 12: height, width, \_ = image.shape

- Dentro de la función `flood\_fill`: Desempaqueta las dimensiones del array `image`. `image.shape` devuelve una tupla (alto, ancho, número de canales). Estas dimensiones se asignan a las variables `height` y `width`. El tercer valor (número de canales de color) se asigna a `\_`, una convención para indicar que no se usará.

Línea 13: old\_color = image[y, x].copy()

- Dentro de la función `flood\_fill`: Accede al píxel en las coordenadas `(y, x)` del array `image` (Numpy usa indexación fila, columna, que corresponde a y, x) para obtener su color original. Se utiliza `.copy()` para asegurar que `old\_color` sea una copia independiente y no una referencia al píxel original en el array.

Línea 14: if np.array\_equal(old\_color, new\_color):

- Dentro de la función `flood\_fill`: Comprueba si el color original (`old\_color`) del píxel semilla ya es igual al color de relleno deseado (`new\_color`) usando `np.array\_equal` para comparar los arrays de color.

Línea 15: return

- Dentro de la función `flood\_fill`: Si los colores son iguales, significa que el área ya está rellena (o el punto inicial no pertenece al área a rellenar con un color diferente), por lo que la función termina inmediatamente sin hacer nada más. Pertenece al bloque `if`.

Línea 16: stack = [(x, y)]

- Dentro de la función `flood\_fill`: Inicializa una lista llamada `stack` que actuará como una pila (stack) para el algoritmo de relleno. Se añade la coordenada inicial `(x, y)` como el primer elemento a procesar.

Línea 17: while stack:

- Dentro de la función `flood\_fill`: Inicia un bucle `while` que continuará mientras la `stack` no esté vacía. Este es el bucle principal del algoritmo flood fill.

Línea 18: cx, cy = stack.pop()

- Dentro del bucle `while`: Extrae (y elimina) el último elemento (una tupla de coordenadas `(x, y)`) de la `stack` y desempaqueta sus valores en las variables `cx` (coordenada x actual) y `cy` (coordenada y actual).

Línea 19: if 0 <= cx < width and 0 <= cy < height and np.array\_equal(image[cy, cx], old\_color):

- Dentro del bucle `while`: Inicia una condición `if` para verificar si el píxel actual `(cx, cy)` debe ser procesado.

- `0 <= cx < width`: Comprueba si la coordenada x está dentro de los límites horizontales de la imagen.

- `and 0 <= cy < height`: Comprueba si la coordenada y está dentro de los límites verticales de la imagen.

- `and np.array\_equal(image[cy, cx], old\_color)`: Comprueba si el color del píxel actual `image[cy, cx]` es igual al `old\_color` original del área que se está rellenando. Solo se rellenan píxeles que cumplen las tres condiciones.

Línea 20: image[cy, cx] = new\_color

- Dentro del bloque `if`: Si todas las condiciones de la línea anterior son verdaderas, se cambia el color del píxel actual `image[cy, cx]` al `new\_color`.

Línea 21: stack.append((cx + 1, cy))

- Dentro del bloque `if`: Añade las coordenadas del píxel vecino de la derecha `(cx + 1, cy)` a la `stack` para que sea procesado posteriormente.

Línea 22: stack.append((cx - 1, cy))

- Dentro del bloque `if`: Añade las coordenadas del píxel vecino de la izquierda `(cx - 1, cy)` a la `stack`.

Línea 23: stack.append((cx, cy + 1))

- Dentro del bloque `if`: Añade las coordenadas del píxel vecino de abajo `(cx, cy + 1)` a la `stack`.

Línea 24: stack.append((cx, cy - 1))

- Dentro del bloque `if`: Añade las coordenadas del píxel vecino de arriba `(cx, cy - 1)` a la `stack`.

Línea 25: (En blanco)

- Línea en blanco: Separa la definición de la función `flood\_fill` del código principal del script.

Línea 26: # Dimensiones de la imagen

- Comentario que indica que las siguientes líneas definirán el tamaño de la imagen a crear.

Línea 27: width, height = 300, 300

- Define dos variables, `width` y `height`, y les asigna el valor 300 a ambas. Estas determinarán las dimensiones de la imagen.

Línea 28: (En blanco)

- Línea en blanco: Separa la definición de dimensiones de la creación de la imagen.

Línea 29: # Crear una imagen en blanco (fondo blanco)

- Comentario que explica el propósito de la siguiente línea.

Línea 30: img = Image.new("RGB", (width, height), "white")

- Crea un nuevo objeto de imagen `Image` usando `Image.new()`.

- `"RGB"`: Especifica el modo de color de la imagen (Rojo, Verde, Azul).

- `(width, height)`: Proporciona las dimensiones de la imagen como una tupla.

- `"white"`: Establece el color de fondo inicial de la imagen como blanco.

- El objeto imagen resultante se asigna a la variable `img`.

Línea 31: draw = ImageDraw.Draw(img)

- Crea un objeto `ImageDraw` asociado con la imagen `img`. Este objeto `draw` proporciona métodos para dibujar sobre la imagen `img`.

Línea 32: (En blanco)

- Línea en blanco: Separa la creación de la imagen/dibujo de la definición de los parámetros del hexágono.

Línea 33: # Parámetros del hexágono regular

- Comentario que indica que las siguientes líneas definirán las propiedades geométricas del hexágono a dibujar.

Línea 34: center = (150, 150)

- Define una tupla `center` que almacena las coordenadas (x, y) del centro del hexágono, que coincide con el centro de la imagen (300/2, 300/2).

Línea 35: radius = 100

- Define una variable `radius` y le asigna el valor 100. Este es el radio del círculo circunscrito al hexágono (la distancia desde el centro a cada vértice).

Línea 36: num\_lados = 6

- Define una variable `num\_lados` y le asigna el valor 6, especificando que se dibujará un hexágono.

Línea 37: (En blanco)

- Línea en blanco: Separa la definición de parámetros del cálculo de los vértices.

Línea 38: # Calcular las coordenadas de los vértices del hexágono

- Comentario que explica el propósito del siguiente bloque de código.

Línea 39: polygon\_points = []

- Inicializa una lista vacía llamada `polygon\_points` que almacenará las coordenadas de los vértices del hexágono.

Línea 40: for i in range(num\_lados):

- Inicia un bucle `for` que iterará `num\_lados` (6) veces, con la variable `i` tomando los valores de 0 a 5.

Línea 41: angle = 2 \* math.pi \* i / num\_lados

- Dentro del bucle `for`: Calcula el ángulo (en radianes) para el vértice actual `i`. Divide un círculo completo (`2 \* math.pi`) en `num\_lados` partes iguales.

Línea 42: x = center[0] + radius \* math.cos(angle)

- Dentro del bucle `for`: Calcula la coordenada x del vértice actual usando trigonometría. `math.cos(angle)` da la componente x en un círculo unitario, se escala por `radius` y se desplaza por la coordenada x del `center`.

Línea 43: y = center[1] + radius \* math.sin(angle)

- Dentro del bucle `for`: Calcula la coordenada y del vértice actual de forma análoga, usando `math.sin(angle)` para la componente y.

Línea 44: polygon\_points.append((x, y))

- Dentro del bucle `for`: Añade la tupla de coordenadas `(x, y)` calculada a la lista `polygon\_points`.

Línea 45: (En blanco)

- Línea en blanco: Separa el cálculo de vértices del dibujo del contorno.

Línea 46: # Dibujar el contorno del hexágono (en negro)

- Comentario que explica la acción de la siguiente línea.

Línea 47: draw.line(polygon\_points + [polygon\_points[0]], fill='black', width=2)

- Llama al método `line()` del objeto `draw`.

- `polygon\_points + [polygon\_points[0]]`: Se le pasa la lista de puntos de los vértices (`polygon\_points`) concatenada con una lista que contiene el primer punto (`polygon\_points[0]`). Esto asegura que se dibuje una línea desde el último vértice de vuelta al primero, cerrando el hexágono.

- `fill='black'`: Establece el color de la línea como negro.

- `width=2`: Establece el grosor de la línea en 2 píxeles.

Línea 48: (En blanco)

- Línea en blanco: Separa el dibujo del contorno de la conversión a array NumPy.

Línea 49: # Convertir la imagen a un array numpy para manipulación

- Comentario que explica la necesidad de convertir la imagen de formato PIL a formato NumPy.

Línea 50: img\_array = np.array(img)

- Convierte el objeto `Image` (`img`) a un array NumPy usando `np.array()`. Cada píxel se representará como un array [R, G, B]. El array resultante se asigna a `img\_array`.

Línea 51: (En blanco)

- Línea en blanco: Separa la conversión a array de la selección del punto semilla.

Línea 52: # Seleccionar un punto semilla dentro del hexágono

- Comentario que explica la selección del punto de inicio para el flood fill.

Línea 53: seed\_x, seed\_y = center # Se utiliza el centro del hexágono

- Desempaqueta la tupla `center` (que contiene las coordenadas (150, 150)) en las variables `seed\_x` y `seed\_y`. Estas serán las coordenadas del píxel inicial para el algoritmo `flood\_fill`.

- El comentario `# Se utiliza el centro del hexágono` aclara la elección del punto semilla.

Línea 54: (En blanco)

- Línea en blanco: Separa la selección del punto semilla de la definición del color de relleno.

Línea 55: # Definir el color de relleno (rojo)

- Comentario que explica qué color se usará para rellenar. Nota: El comentario dice "rojo", pero el código define amarillo. \*Actualización: El código proporcionado en el prompt SÍ define rojo (255, 0, 0). Seguiré con el código tal cual.\*

Línea 56: new\_color = np.array([255, 0, 0], dtype=np.uint8)

- Crea un array NumPy llamado `new\_color` que representa el color de relleno.

- `[255, 0, 0]`: Define el color como Rojo (R=255, G=0, B=0).

- `dtype=np.uint8`: Especifica el tipo de dato de los elementos del array como enteros sin signo de 8 bits, que es el tipo estándar para valores de color de píxeles (0-255).

Línea 57: (En blanco)

- Línea en blanco: Separa la definición del color de la llamada a la función flood fill.

Línea 58: # Aplicar flood fill al área interna del hexágono

- Comentario que explica la acción de la siguiente línea.

Línea 59: flood\_fill(img\_array, seed\_x, seed\_y, new\_color)

- Llama a la función `flood\_fill` definida anteriormente.

- `img\_array`: Pasa el array NumPy de la imagen como el primer argumento. La función modificará este array directamente.

- `seed\_x`, `seed\_y`: Pasa las coordenadas del punto semilla (el centro).

- `new\_color`: Pasa el color de relleno (rojo).

Línea 60: (En blanco)

- Línea en blanco: Separa la ejecución del flood fill de la visualización del resultado.

Línea 61: # Visualizar el resultado

- Comentario que indica que las siguientes líneas mostrarán la imagen resultante.

Línea 62: plt.figure(figsize=(6, 6))

- Llama a `plt.figure()` para crear una nueva figura de Matplotlib donde se mostrará la imagen.

- `figsize=(6, 6)`: Establece el tamaño de la figura en 6x6 pulgadas.

Línea 63: plt.imshow(img\_array)

- Llama a `plt.imshow()` para mostrar los datos del array `img\_array` como una imagen dentro de la figura actual.

Línea 64: plt.title("Relleno de Hexágono Regular con Flood Fill")

- Llama a `plt.title()` para establecer el título del gráfico.

Línea 65: plt.axis('off')

- Llama a `plt.axis('off')` para desactivar la visualización de los ejes (x e y) y sus etiquetas, mostrando solo la imagen.

Línea 66: plt.show()

- Llama a `plt.show()` para abrir la ventana de Matplotlib y mostrar la figura con la imagen rellena. El script se pausará aquí hasta que la ventana se cierre.

Resumen del Código

Este código Python utiliza las bibliotecas Pillow (PIL) y NumPy para generar una imagen y rellenar una forma dentro de ella, y Matplotlib para mostrar el resultado.

1. **Importaciones y Función flood\_fill:** Importa las bibliotecas necesarias (numpy, PIL.Image, PIL.ImageDraw, matplotlib.pyplot, math). Define una función flood\_fill que implementa el algoritmo de relleno recursivo (usando una pila explícita) para cambiar el color de un área conectada de píxeles a partir de un punto semilla, operando sobre un array NumPy de la imagen.
2. **Creación de la Imagen Base:** Crea una imagen en blanco de 300x300 píxeles usando Pillow.
3. **Cálculo y Dibujo del Hexágono:** Calcula las coordenadas de los vértices de un hexágono regular centrado en la imagen usando funciones trigonométricas (math.cos, math.sin). Dibuja el contorno de este hexágono en negro sobre la imagen usando ImageDraw.
4. **Conversión y Preparación:** Convierte la imagen de Pillow a un array NumPy para poder aplicar el flood\_fill. Define el punto semilla como el centro de la imagen y el color de relleno como rojo ([255, 0, 0]).
5. **Aplicación del Relleno:** Llama a la función flood\_fill pasándole el array de la imagen, el punto semilla (el centro del hexágono, que inicialmente es blanco) y el color rojo. La función modifica el array img\_array, cambiando todos los píxeles blancos conectados al centro (es decir, el interior del hexágono) a rojo.
6. **Visualización:** Utiliza Matplotlib para mostrar el img\_array modificado como una imagen, con un título apropiado y sin ejes visibles. El resultado final es una imagen de un hexágono con borde negro relleno de rojo sobre un fondo blanco.

Forma, Polígono

El contenido generado por IA puede ser incorrecto.