```
# Project 10: Discrete dynamical systems, linear transformations of the plane,
import numpy as np
import matplotlib.pyplot as plt


print("\nSubtask 1\n")
# Generate linearly spaced values
t = np.linspace(0, 2 * np.pi, 4)
# Remove the fourth element
t = np.delete(t, 3)
# Define the vertices of the equilateral triangle
v = np.array([np.cos(t), np.sin(t)])
```

Subtask 1

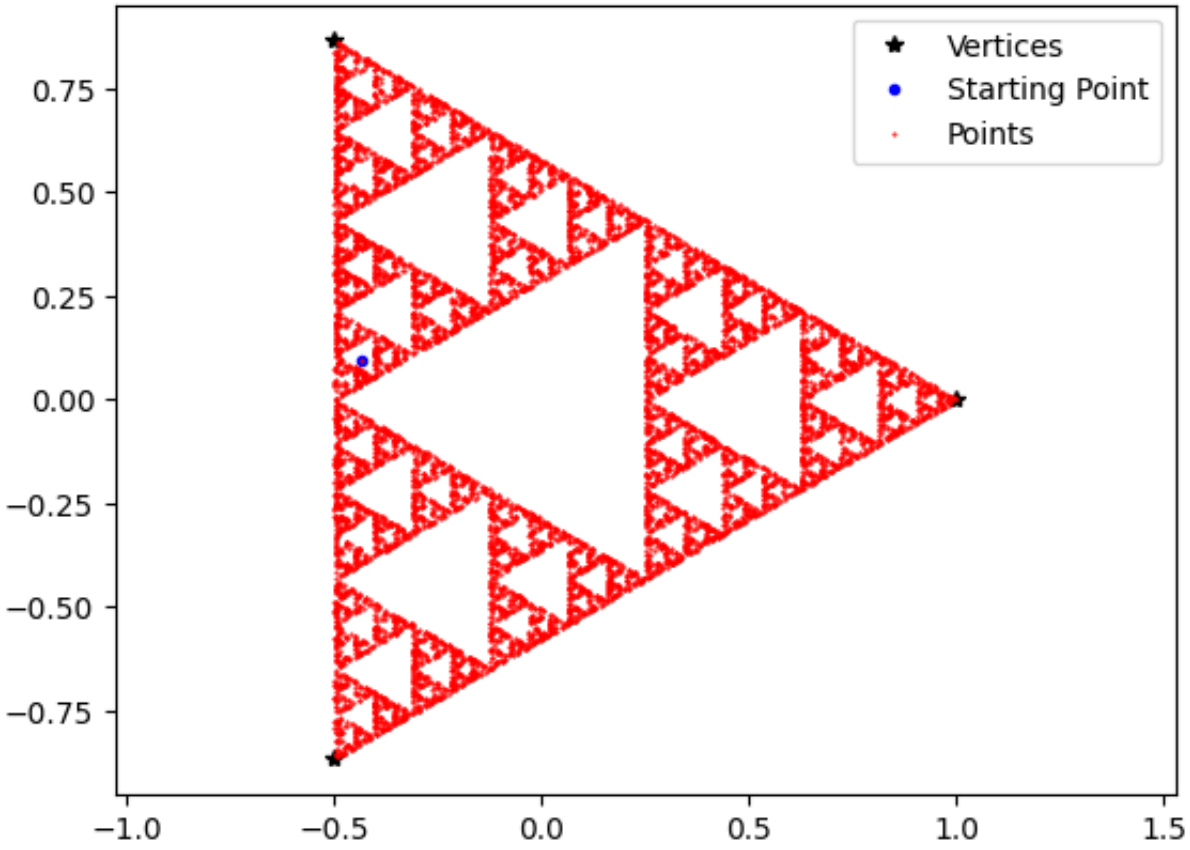
```
print("\nSubtask 2\n")
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point with random values between -0.5 and 0.5
x = np.random.rand(2, 1) - 0.5
```

Subtask 2

```
print("\nSubtask 3\n")
# Define the vertices of the triangle
t = np.linspace(0, 2 * np.pi, 4)[:-1] # Generate t and remove the last element
v = np.array([np.cos(t), np.sin(t)])
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num))
# Initial point
current_point = x
# Iterative process
for i in range(Num):
    # Store the current point
    points[:, i] = current_point.flatten()
    # Choose a random vertex
```

```
    vertex = v[:, np.random.choice(3)]
    # Apply the linear transformation and update the point
    current_point = T @ (current_point + vertex.reshape(2, 1))
# Plot the vertices of the triangle
plt.plot(v[0, :], v[1, :], 'k*', label='Vertices')
# Plot the starting point
plt.plot(x[0, 0], x[1, 0], 'b.', label='Starting Point')
# Plot all the points obtained during the iterations
plt.plot(points[0, :], points[1, :], 'r.', markersize=1, label='Points')
plt.axis('equal')
plt.legend()
plt.show()
```

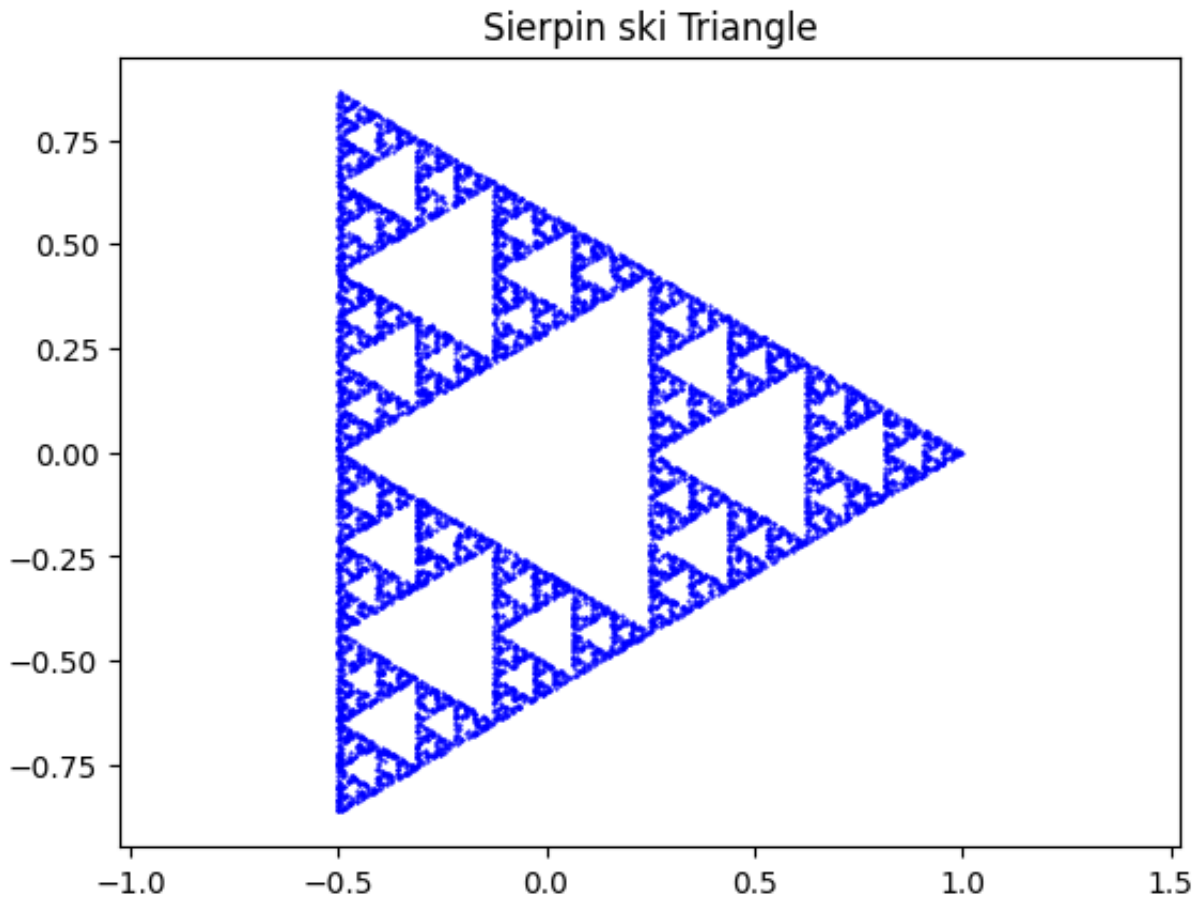
```
Subtask 3
```




```
print("\nSubtask 4\n")
# Define the vertices of the triangle
t = np.linspace(0, 2 * np.pi, 4)[:-1] # Generate t and remove the last element
v = np.array([np.cos(t), np.sin(t)])
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
```

```
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 3) # Random integer from 0 to 2
    # Perform the transformation
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - v[:, k]) + v[:, k]
    points[:, j + 1] = transformed_point
# Plot the points
plt.plot(points[0, :], points[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Sierpin ski Triangle')
plt.show()
```

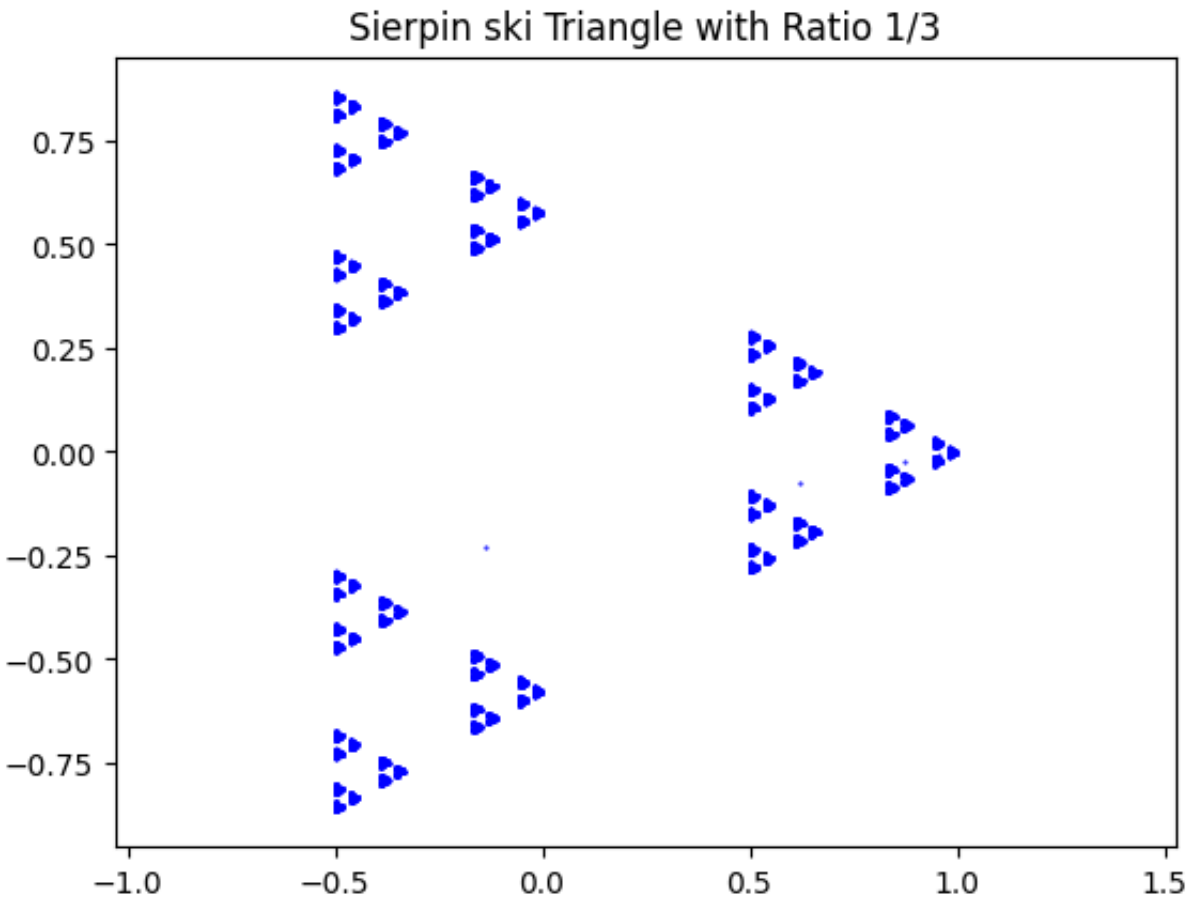
```
Subtask 4
```




```
print("\nSubtask 5\n")
# Define the vertices of the triangle
t = np.linspace(0, 2 * np.pi, 4)[:-1] # Generate t and remove the last element
```

```
v = np.array([np.cos(t), np.sin(t)])
# Define the new linear transformation matrix with a ratio of 1/3
T = np.array([[1/3, 0], [0, 1/3]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 3) # Random integer from 0 to 2
    # Perform the transformation
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - v[:, k]) + v[:, k]
    points[:, j + 1] = transformed_point
# Plot the points
plt.plot(points[0, :], points[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Sierpin ski Triangle with Ratio 1/3')
plt.show()
```

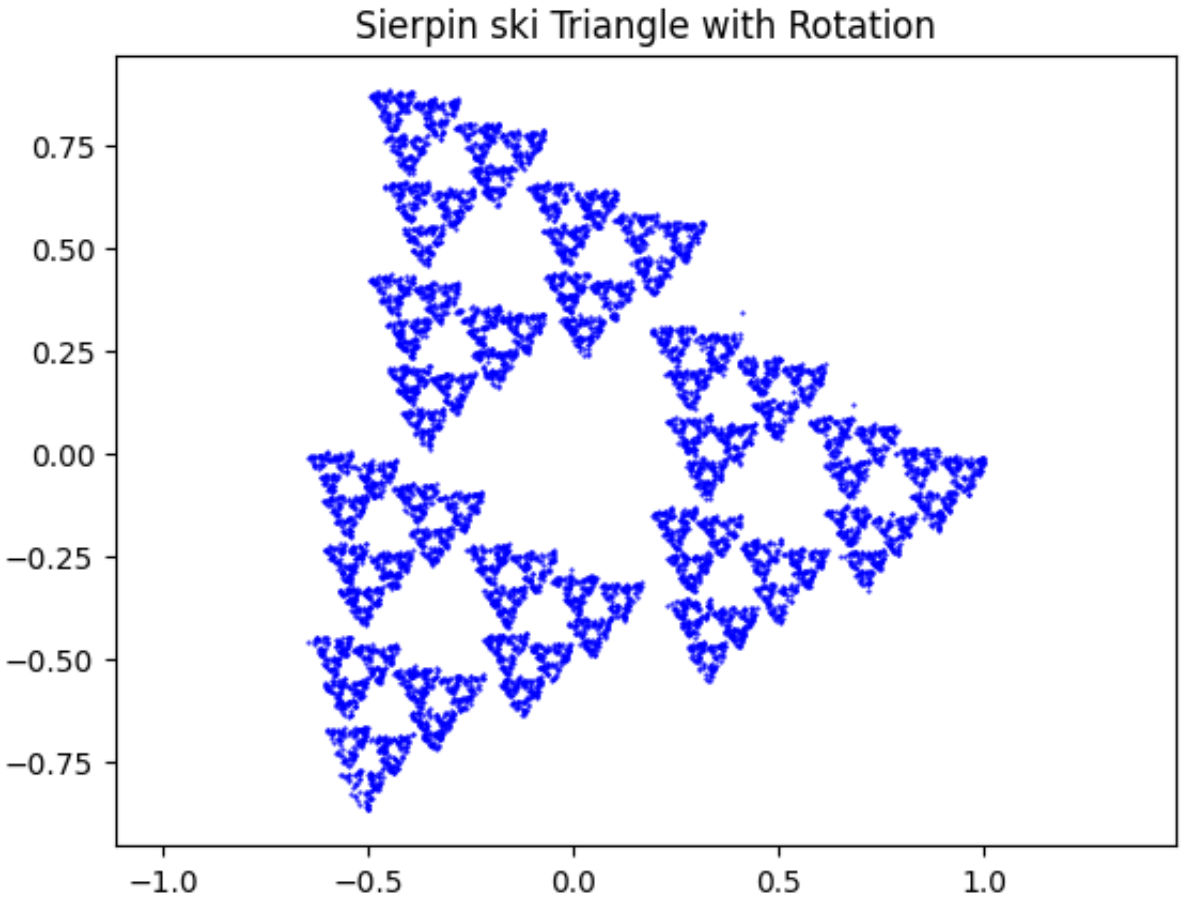
```
Subtask 5
```




```
print("\nSubtask 6\n")
# Define the vertices of the triangle
t = np.linspace(0, 2 * np.pi, 4)[:-1] # Generate t and remove the last element
v = np.array([np.cos(t), np.sin(t)])
# Define the rotation angle and transformation matrix T
theta = np.pi / 18
T = 0.5 * np.array([[np.cos(theta), -np.sin(theta)],
[np.sin(theta), np.cos(theta)]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 3) # Random integer from 0 to 2
    # Perform the transformation
```

```
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - v[:, k]) + v[:, k]
    points[:, j + 1] = transformed_point
# Plot the points
plt.plot(points[0, :], points[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Sierpin ski Triangle with Rotation')
plt.show()
```

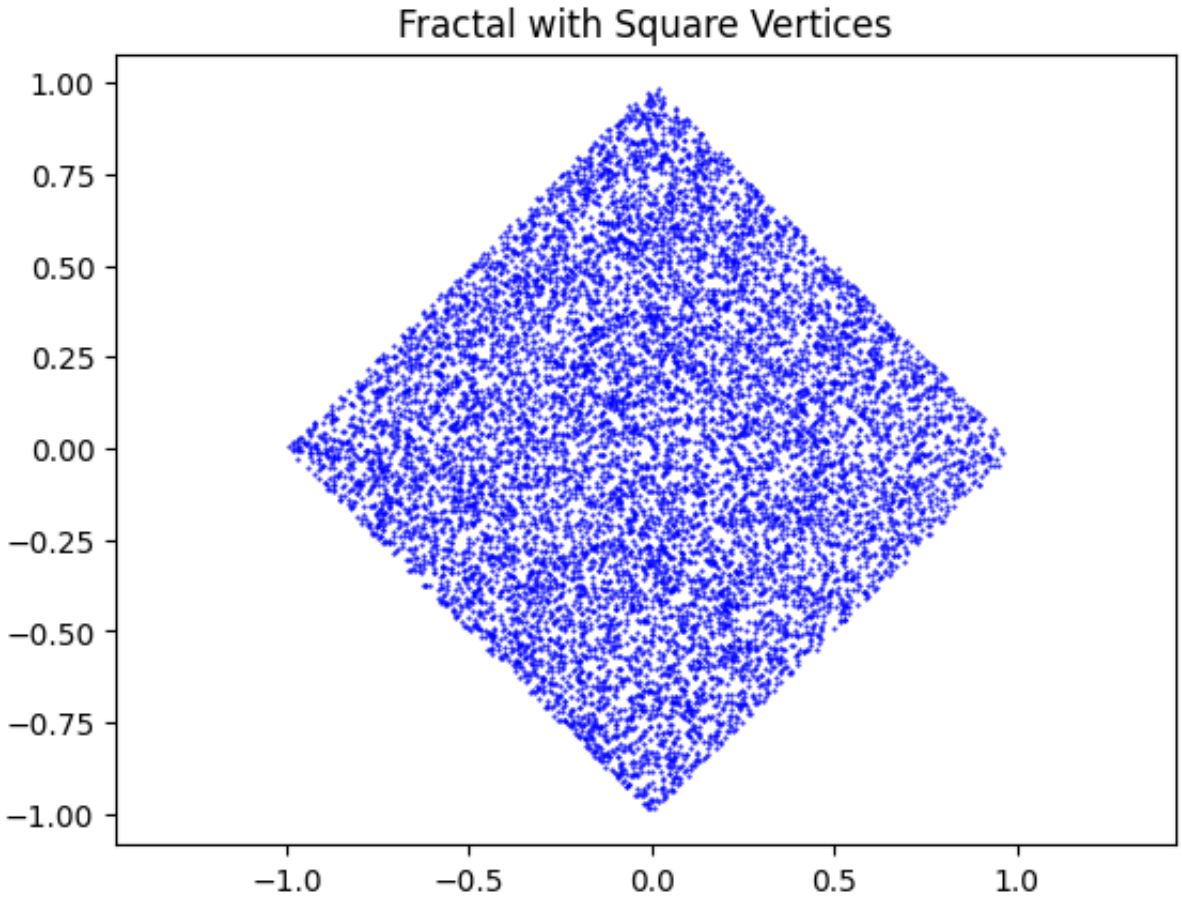
```
Subtask 6
```




```
print("\nSubtask 7\n")
# Define the vertices of the square
t = np.linspace(0, 2 * np.pi, 5)[:-1] # Generate t and remove the last element
v = np.array([np.cos(t), np.sin(t)])
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num + 1))
```

```
# Initial point
points[:, 0] = x.flatten()
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 4) # Random integer from 0 to 3 (for 4 vertices)
    # Perform the transformation
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - v[:, k]) + v[:, k]
    points[:, j + 1] = transformed_point
# Plot the points
plt.figure()
plt.plot(points[0, :], points[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Fractal with Square Vertices')
plt.show()
```

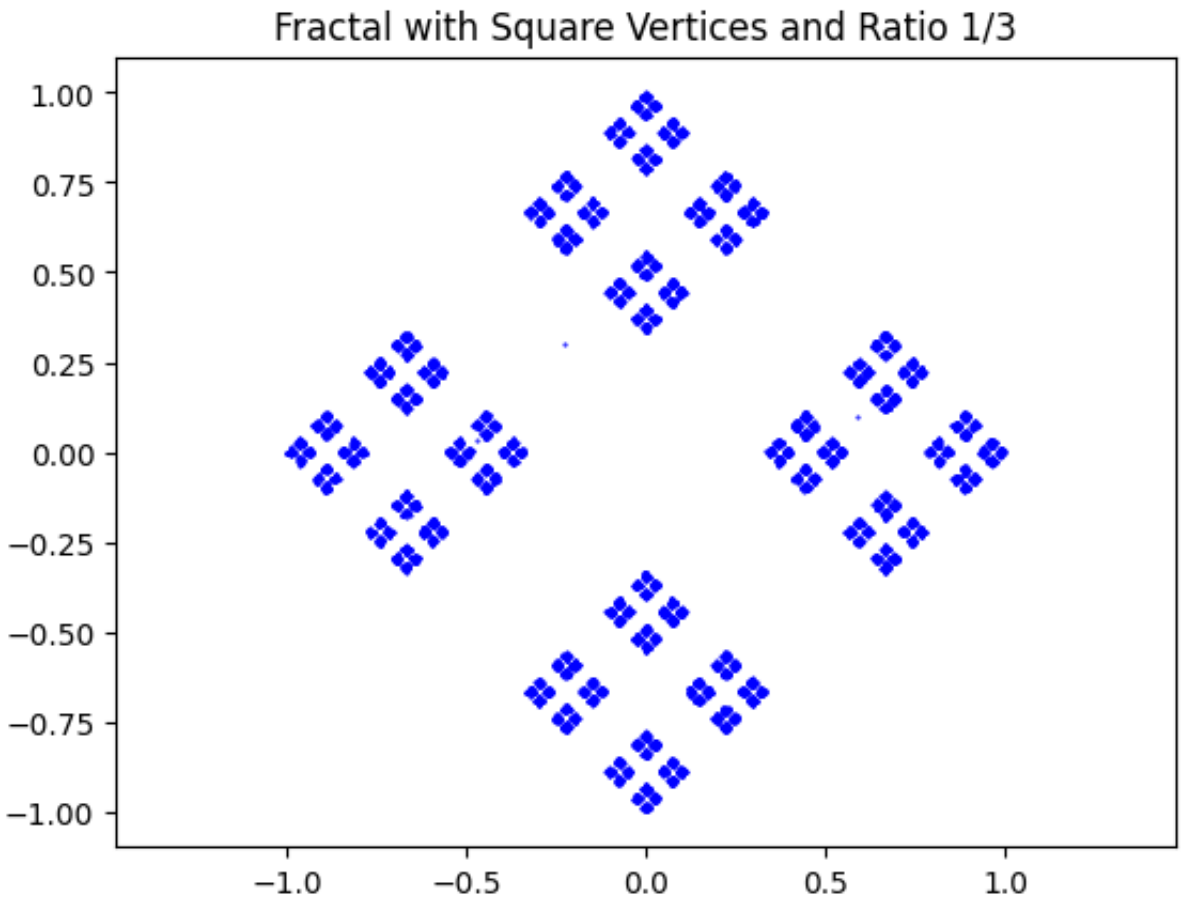
```
Subtask 7
```




```
print("\nSubtask 8\n")
# Define the vertices of the square
t = np.linspace(0, 2 * np.pi, 5)[:-1] # Generate t and remove the last element
v = np.array([np.cos(t), np.sin(t)])
# Define the new linear transformation matrix with a ratio of 1/3
```

```
T = np.array([[1/3, 0], [0, 1/3]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 10000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 4) # Random integer from 0 to 3 (for 4 vertices)
    # Perform the transformation
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - v[:, k]) + v[:, k]
    points[:, j + 1] = transformed_point
# Plot the points
plt.figure()
plt.plot(points[0, :], points[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Fractal with Square Vertices and Ratio 1/3')
plt.show()
```

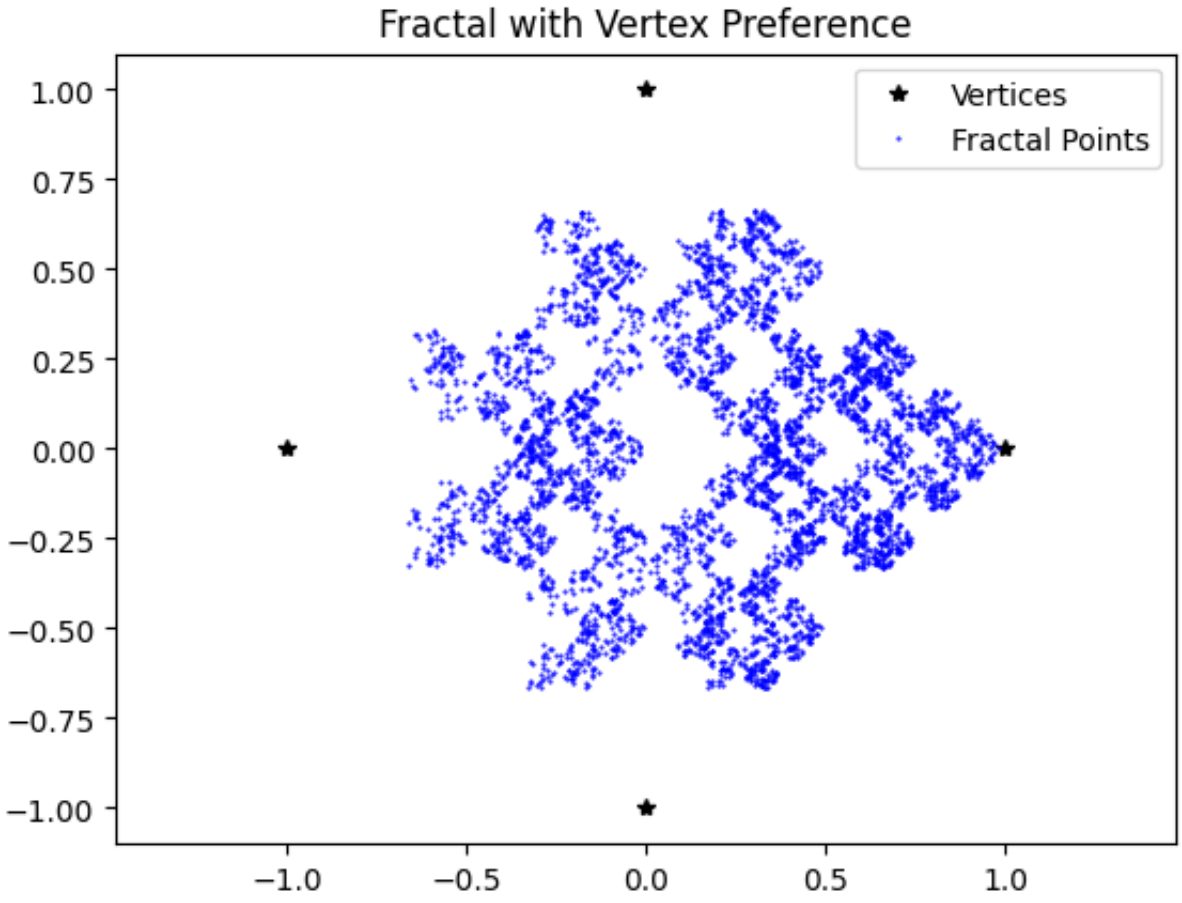
```
Subtask 8
```




```
print("\nSubtask 9\n")
# Define the vertices of the square using complex exponentials
t = np.linspace(0, 2 * np.pi, 5)[:-1] # Generate t and remove the last element
v = np.exp(1j * t) # Complex exponential for the vertices
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 5000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Variable to keep track of the previous vertex index
k1 = -1
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 4) # Random integer from 0 to 3
    # Ensure the same vertex is not selected twice in a row
```

```
    if k >= k1:
        k = (k + 1) % 4 # Move to the next vertex if k >= k1
    # Get the current vertex
    w = np.array([np.real(v[k]), np.imag(v[k])])
    # Perform the transformation
    current_point = points[:, j] # Get the current point
    transformed_point = T @ (current_point - w) + w
    points[:, j + 1] = transformed_point
    # Update the previous vertex index
    k1 = k
# Plot the vertices and the points
plt.figure()
plt.plot(np.real(v), np.imag(v), 'k*', label='Vertices') # Plot vertices
plt.plot(points[0, :], points[1, :], 'b.', markersize=1, label='Fractal Points'
plt.axis('equal')
plt.title('Fractal with Vertex Preference')
plt.legend()
plt.show()
```

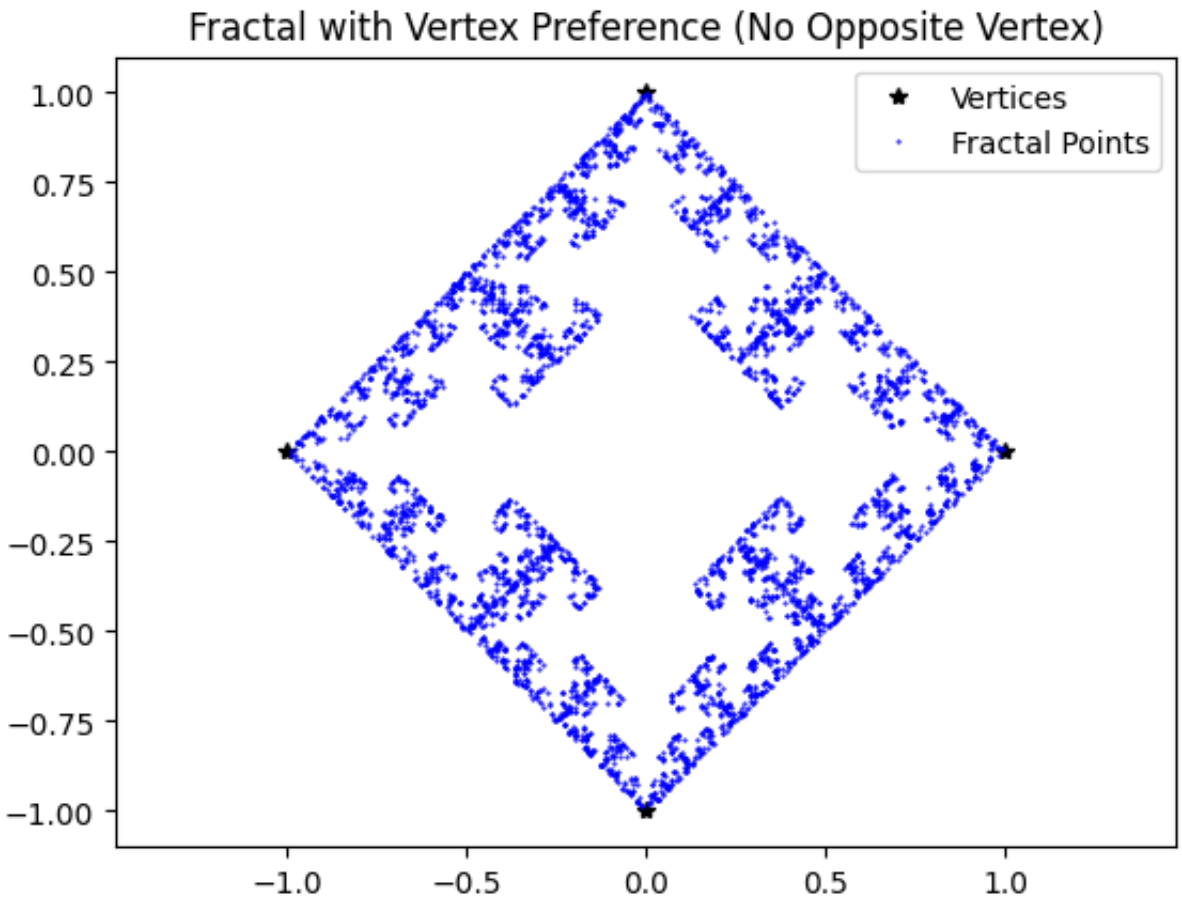
Subtask 9




```
print("\nSubtask 10\n")
# Define the vertices of the square using complex exponentials
```

```
t = np.linspace(0, 2 * np.pi, 5)[:-1] # Generate t and remove the last element
v = np.exp(1j * t) # Complex exponential for the vertices
# Define the linear transformation matrix
T = np.array([[0.5, 0], [0, 0.5]])
# Define the starting point
x = np.random.rand(2, 1) - 0.5
# Number of iterations
Num = 5000
# Create an array to store all points
points = np.zeros((2, Num + 1))
# Initial point
points[:, 0] = x.flatten()
# Variable to keep track of the previous vertex index
k1 = 0
w = np.array([np.real(v[k1]), np.imag(v[k1])])
# Iterative process
for j in range(Num):
    k = np.random.randint(0, 4) # Random integer from 0 to 3
    # Ensure the new vertex is not opposite to the previous one
    if (k != (k1 + 2) % 4) and ((k1 != (k + 2) % 4)):
        w = np.array([np.real(v[k]), np.imag(v[k])])
        transformed_point = T @ (points[:, j] - w) + w
        points[:, j + 1] = transformed_point
        k1 = k # Update the previous vertex index
    else:
        points[:, j + 1] = points[:, j] # If opposite, repeat the current point
# Plot the vertices and the points
plt.figure()
plt.plot(np.real(v), np.imag(v), 'k*', label='Vertices') # Plot vertices
plt.plot(points[0, :], points[1, :], 'b.', markersize=1, label='Fractal Points'
plt.axis('equal')
plt.title('Fractal with Vertex Preference (No Opposite Vertex)')
plt.legend()
plt.show()
```

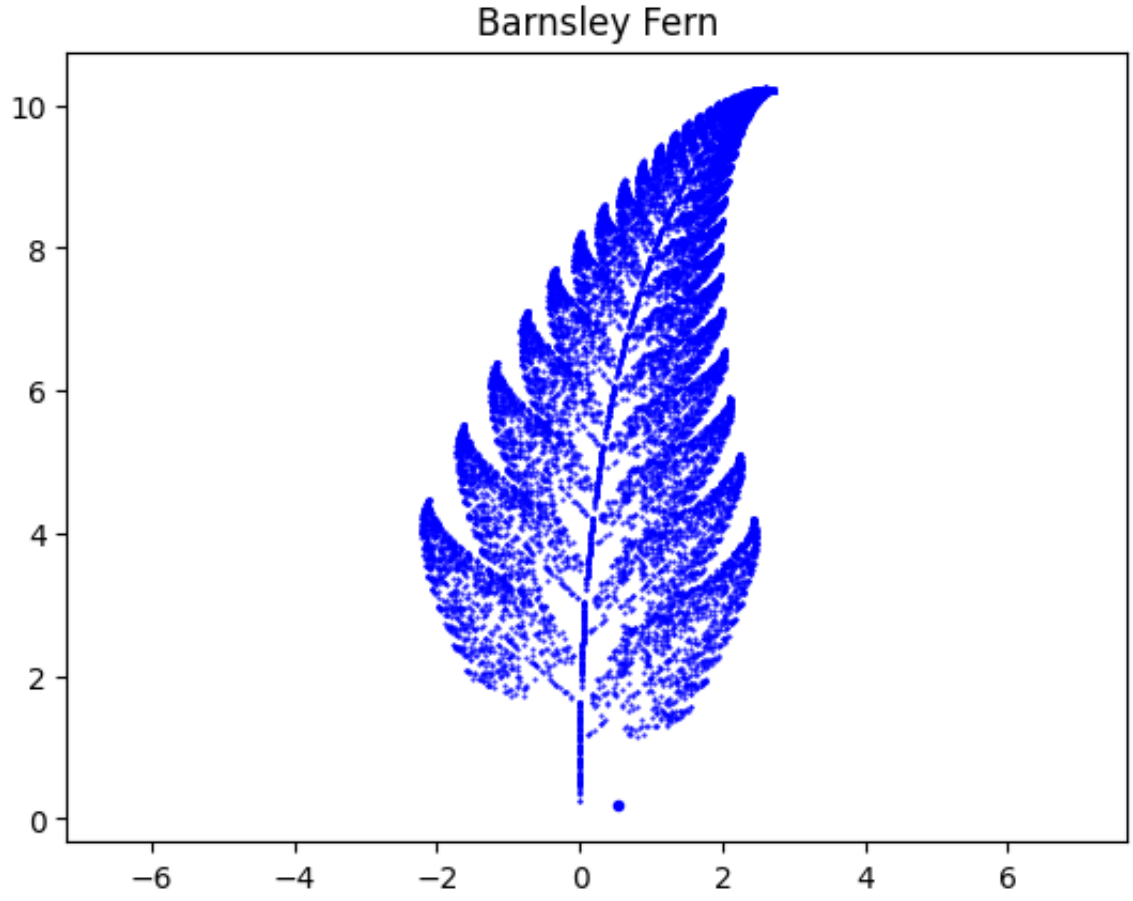
```
Subtask 10
```




```
print("\nSubtask 11\n")
# Define the transformation matrices and translation vectors
T1 = np.array([[0.85, 0.04], [-0.04, 0.85]])
T2 = np.array([[-0.15, 0.28], [0.26, 0.24]])
T3 = np.array([[0.2, -0.26], [0.23, 0.22]])
T4 = np.array([[0, 0], [0, 0.16]])
Q1 = np.array([0, 1.64])
Q2 = np.array([-0.028, 1.05])
Q3 = np.array([0, 1.6])
Q4 = np.array([0, 0])
P1 = 0.85
P2 = 0.07
P3 = 0.07
P4 = 0.01
Num = 15000
# Initialize the starting point
x = np.zeros((2, Num + 1))
x[:, 0] = np.random.rand(2) # Starting point
# Plot the initial point
plt.figure()
```

```
plt.plot(x[0, 0], x[1, 0], 'b.')
# Iterative process to generate the fern
for j in range(Num):
    r = np.random.rand()
    if r <= P1:
        x[:, j + 1] = T1 @ x[:, j] + Q1
    elif r <= P1 + P2:
        x[:, j + 1] = T2 @ x[:, j] + Q2
    elif r <= P1 + P2 + P3:
        x[:, j + 1] = T3 @ x[:, j] + Q3
    else:
        x[:, j + 1] = T4 @ x[:, j] + Q4
# Plot the fractal
plt.plot(x[0, :], x[1, :], 'b.', markersize=1)
plt.axis('equal')
plt.title('Barnsley Fern')
plt.show()
```

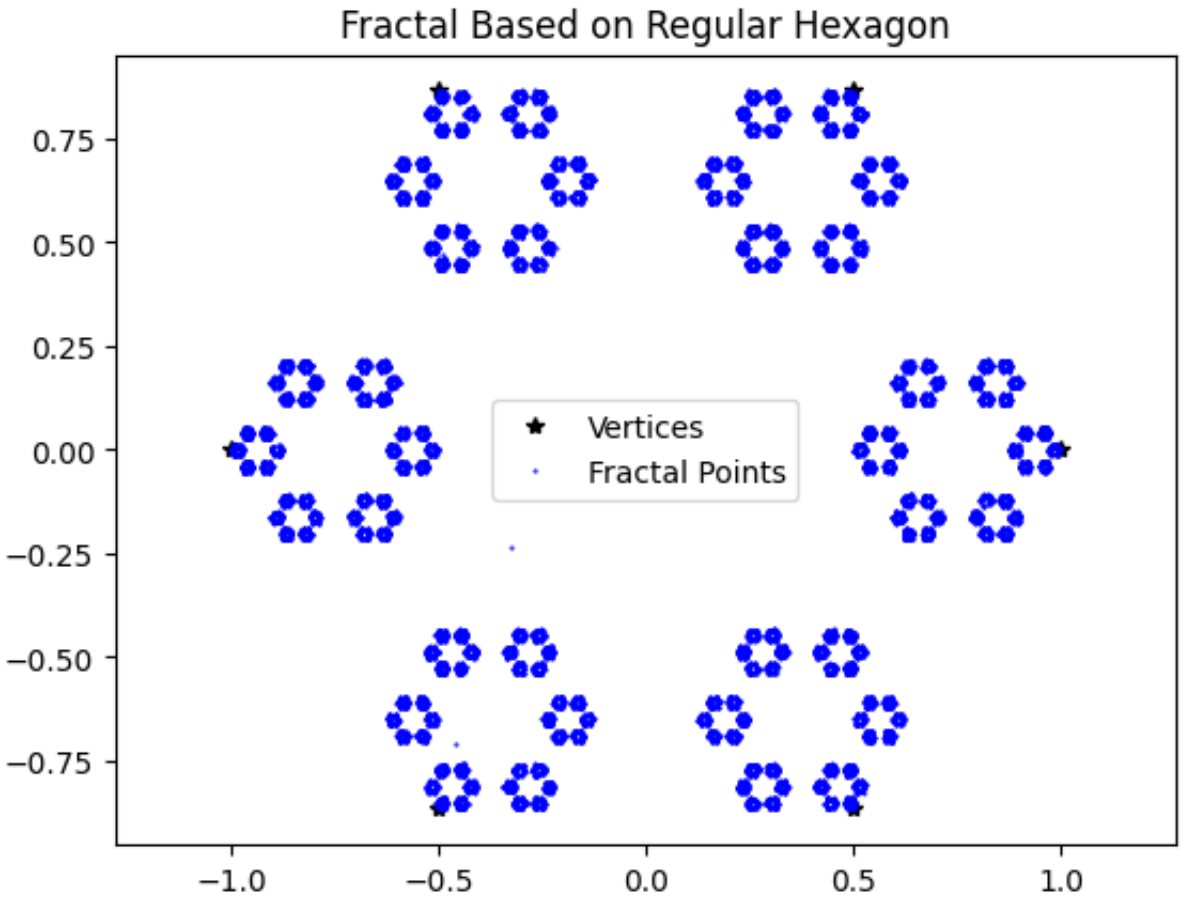
```
Subtask 11
```




```
print("\nSubtask 12\n")
# Define the vertices of a regular hexagon
t = np.linspace(0, 2 * np.pi, 7)[:-1] # Generate 6 vertices and remove the last
```

```
v = np.array([np.cos(t), np.sin(t)]) # Calculate vertices
# Define the linear transformation matrix
T = np.array([[0.25, 0], [0, 0.25]])
# Define the number of iterations
Num = 15000
# Initialize the starting point
x = np.zeros((2, Num + 1))
x[:, 0] = np.random.rand(2) - 0.5 # Random starting point
# Create an array to store all points
points = np.zeros((2, Num + 1))
points[:, 0] = x[:, 0]
# Iterative process to generate the fractal
for j in range(Num):
    k = np.random.randint(0, 6) # Randomly choose one of the 6 vertices
    points[:, j + 1] = T @ (points[:, j] - v[:, k]) + v[:, k]
# Plot the vertices and the points
plt.figure()
plt.plot(v[0, :], v[1, :], 'k*', label='Vertices') # Plot vertices of hexagon
plt.plot(points[0, :], points[1, :], 'b.', markersize=1, label='Fractal Points'
points
plt.axis('equal')
plt.title('Fractal Based on Regular Hexagon')
plt.legend()
plt.show()
```

```
Subtask 12
```




```
print("\nSubtask 13\n")
# Define the vertices of a regular pentagon
t = np.linspace(0, 2 * np.pi, 6)[:-1] # Generate 5 vertices
v = np.array([np.cos(t), np.sin(t)]) # Calculate vertices
# Define the linear transformation matrix
T = np.array([[2/5, 0], [0, 2/5]])
# Define the number of iterations
Num = 10000
# Initialize the starting point
x = np.zeros((2, Num + 1))
x[:, 0] = np.random.rand(2) - 0.5 # Random starting point
# Create an array to store all points
points = np.zeros((2, Num + 1))
points[:, 0] = x[:, 0]
# Initialize previous vertex index
prev_vertex_index = np.random.randint(0, 5)
# Iterative process to generate the fractal
for j in range(Num):
    # Randomly choose one of the 5 vertices but not the same as the previous one
    current_vertex_index = prev_vertex_index
    while current_vertex_index == prev_vertex_index:
```

```
        while current_vertex_index == prev_vertex_index:
            current_vertex_index = np.random.randint(0, 5)
        # Apply the transformation
        w = v[:, current_vertex_index]
        points[:, j + 1] = T @ (points[:, j] - w) + w
        # Update the previous vertex index
        prev_vertex_index = current_vertex_index
# Plot the vertices and the points
plt.figure()
plt.plot(v[0, :], v[1, :], 'k*', label='Vertices') # Plot vertices of pentagon
plt.plot(points[0, :], points[1, :], 'b.', markersize=1, label='Fractal Points')
points
plt.axis('equal')
plt.title('Fractal Based on Regular Pentagon with Vertex Restriction')
plt.legend()
plt.show()
```

Subtask 13