What This Program Does

- **Normalize Pixel Values**: Convert each pixel value from 0–255 to 0–1.
- **Rate Coding**: Use the normalized value to generate spikes (0 or 1) for each time step based on probability.
- **Spike Visualization**: Show the spike pattern for each time step as a 3×3 grid on the serial monitor.

Code Overview

```
import urandom
import utime

MAX_RATE = 1.0  # 100% firing rate

def rate_code(value, steps=10):
    spikes = []
    for _ in range(steps):
        r = urandom.getrandbits(8) / 256.0
        spikes.append(1 if r < value else 0)
    return spikes

def pattern_to_spikes(pattern, steps=10):
    spike_matrix = []
    for row in pattern:
        spike_row = []
        for val in row:
            norm_val = val / 255.0
            spike_row.append(rate_code(norm_val * MAX_RATE, steps))
        spike_matrix.append(spike_row)
    return spike_matrix

def display_spikes(spike_matrix, step):
    print("Step", step)
    for row in spike_matrix:
        line = ""
```

```
        for spike_seq in row:
            line += "█" if spike_seq[step] == 1 else " "
        print(line)
    print("")

pattern = [
    [255,   0, 255],
    [  0, 255,   0],
    [255,   0, 255]
]

steps = 10
spike_matrix = pattern_to_spikes(pattern, steps)

for t in range(steps):
    display_spikes(spike_matrix, t)
    utime.sleep(0.2)
```

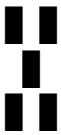
Function Descriptions

- **`rate_code(value, steps)`**
  Creates a spike train (list of 0s and 1s) based on a given value.
  Example: If `value = 0.8`, about 8 spikes will appear in 10 steps.
- **`pattern_to_spikes(pattern, steps)`**
  Converts a 3x3 pattern (with pixel values like 0 or 255) into a 3x3 spike matrix.
  Brighter pixels (255) fire more often; darker pixels (0) rarely fire.
- **`display_spikes(spike_matrix, step)`**
  Shows the spike status at a specific time step in a 3x3 grid.
  "█" = spike (1), " " = no spike (0)

Example Output

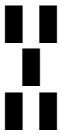
```
vbnet
CopyEdit
```

Step 0



Step 1

Step 2



...