```python
import random
import numpy as np

# CONSTANTS
p = np.array([12, 24, 48, 70, 100, 120])
N = 120
n_trials = 100000


# FUNCTIONS
def generate_random_matrix(n):
    r = np.random.randint(2, size=(n, N))
    r[r == 0] = -1
    return r


def perform_one_trial(pattern_n):
    # Generate weight matrix
    patterns = generate_random_matrix(pattern_n)
    W = (1/N) * np.matmul(patterns.T, patterns)
    # np.fill_diagonal(W, 0)

    # Choose random pattern
    nu_index = random.randint(0, pattern_n - 1)
    chosen_pattern = patterns[nu_index]
    # Choose random neuron
    neuron_index = random.randint(0, N - 1)
    target_neuron_value = chosen_pattern[neuron_index]

    # Feed chosen pattern to network
    selected_weights = np.matrix(W[neuron_index]).T
    new_neuron_value = np.sign(np.dot(chosen_pattern, selected_weights))
    new_neuron_value = new_neuron_value.item()
    if new_neuron_value == 0:
        new_neuron_value = 1
    return new_neuron_value == target_neuron_value


def perform_n_trials(p, n):
    success = 0
    for i in range(n):
        success += perform_one_trial(p)
```

```
    print("Error probability for " + str(p) + " patterns: " + str(1 - success / n))
    return 1 - success / n


# MAIN CODE
pnt_vec = np.vectorize(perform_n_trials)
print(pnt_vec(p, n_trials))

```

```python
import numpy as np
import matplotlib.pyplot as plt

# PATTERNS
x1 = np.array([ [ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1] ])
x2 = np.array([ [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1] ])
x3 = np.array([ [ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1] ])
x4 = np.array([ [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1
```

```
, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1] ])
x5 = np.array([ [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, 1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1] ])

original_shape = x1.shape
patterns = [x1, x2, x3, x4, x5]
fig = plt.figure()
N = original_shape[0]*original_shape[1]
rounds_per_iter = 10000


# RESHAPE
for i, val in enumerate(patterns):
    patterns[i] = np.reshape(val, (N))

# WEIGHT MATRIX
pattern_matrix = np.matrix(patterns)
W = np.matmul(pattern_matrix.T, pattern_matrix)
np.fill_diagonal(W, 0)

def ms_distance(x,y):
    return np.sum(np.square((x - y)))

def draw_digit(x):
    ax1 = fig.add_subplot(121)
    ax1.imshow(x, interpolation='nearest')
```

```python

def feed(x):
    iteration = 0
    running = True

    while running:
        old_x = x.copy()
        for i in range(N):
            # Update all neurons
            neuron_index = i
            selected_weights = np.matrix(W[neuron_index]).T
            new_neuron_value = np.sign(np.dot(x, selected_weights))
            new_neuron_value = new_neuron_value.item()
            if new_neuron_value == 0:
                new_neuron_value = 1
            x[0][neuron_index] = new_neuron_value
        curr_dist = ms_distance(old_x, x)
        print("it. finished : " + str(curr_dist))
        running = (curr_dist != 0)
    x = x.reshape(original_shape)
    draw_digit(x)
    print(x.tolist())


# MAIN
x = np.array([[1, -1, -1, 1, 1, 1, 1, -1, -1, 1], [1, -1, -1, 1, 1, 1, 1, -1, -1, 1], [1, -1, -1, 1, 1, 1, 1, -1, -1, 1], [1, -1, -1, 1, 1, 1, 1, -1, -1, 1], [1, -1, -1, 1, 1, 1, 1, -1, -1, 1], [1, -1, -1, -1, -1, -1, -1, -1, -1, 1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, -1, 1, 1, -1]])
# x = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, -1, -1, -1, -1, 1, 1, 1], [-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1
```

```
, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]])
# x = np.array([[1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, -1, -1, 1, -1, 1, 1, -1], [1, 1, 1, 1, 1, -1, 1, 1, 1, -1], [1, 1, 1, 1, 1, -1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1], [1, -1, -1, -1, -1, 1, -1, 1, 1, -1]])

x = np.reshape(x, (1,N))
feed(x)
plt.show()
```

```
import random
import numpy as np
import matplotlib.pyplot as plt

# CONSTANTS
N = 200
BETA = 2
P = 45
T_N = 200000
T_REPEAT = 100


# FUNCTIONS
def generate_random_matrix(n):
    r = np.random.randint(2, size=(n, N))
    r[r == 0] = -1
    return r


def perform_one_trial():
    # x1 = np.linspace(0, T_N - 1, T_N)
    # x2 = np.array([0.] * T_N)
    # fig = plt.figure()
    # ax = fig.add_subplot(111)
    # line1, = ax.plot(x1, x2, 'r-')
    # plt.ion()
    # plt.ylim([0., 1.1])
    # plt.xlim([0., int(T_N / 200) - 1])
    # plt.show()

    # Generate weight matrix
    patterns = generate_random_matrix(P)
    W = (1/N) * np.matmul(patterns.T, patterns)
    np.fill_diagonal(W, 0)

    # Choose first pattern
    nu_index = 1
    chosen_pattern = patterns[nu_index]
    current_state = chosen_pattern.copy()

    order_sum = 0

    for iteration in range(int(T_N / 200)):
        for neuron_index in range(N):
```

```python

            # Feed chosen pattern to network
            selected_weights = np.matrix(W[neuron_index]).T
            b = np.dot(current_state, selected_weights).item()
            # if b == 0:
            #     b = 1

            b1 = np.exp(-2 * BETA * b)

            p_b = 1 / (1 + b1)
            new_neuron_value = -1
            r = random.uniform(0., 1.0)

            if r < p_b:
                new_neuron_value = 1
            current_state[neuron_index] = new_neuron_value

        sum_t = 1/N * np.sum(current_state * chosen_pattern)
        order_sum = (iteration*order_sum + sum_t) / (iteration + 1)
    #     x2[iteration] = order_sum
    #     line1.set_ydata(x2)
    #     fig.canvas.draw()
    #     fig.canvas.flush_events()
    #
    # input()
    return order_sum


def perform_n_trials():
    x1 = np.linspace(0, T_REPEAT - 1, T_REPEAT)
    x2 = np.array([0.] * T_REPEAT)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    line1, = ax.plot(x1, x2, 'r-')
    plt.ion()
    plt.ylim([0., 1.1])
    plt.xlim([0., T_REPEAT - 1])
    plt.show()
```

```
    order_sum = 0.

    for i in x1:
        a = perform_one_trial()
        x2[int(i)] = a
        line1.set_ydata(x2)
        fig.canvas.draw()
        fig.canvas.flush_events()
        print(a)
        order_sum += a

    print("<m1> = " + str(order_sum / float(T_REPEAT)))
    input() # Block afterwards

# MAIN CODE
perform_n_trials()


```