```python
#!/usr/bin/python3
# Alberto Maria Segre
#
# Copyright 2015, The University of Iowa.  All rights reserved.
# Permission is hereby given to use and reproduce this software
# for non-profit educational purposes only.
from random import random, randint, sample
import matplotlib.pyplot as plt

# Function to roll a weighted die. Returns True with probability p.
# else False.
def rolldie (p):
    '''Returns True with probability p.'''
    return(random() <= p)

# Our infection model is quite simple (see Carrat et al, 2008). People
# are exposed for E days (the incubation period), then infected for I
# additional days (the symptomatic period). Individuals are infectious
# as either E or I.  Carrat et al. (2008) indicate E~2 and I~7 for
# influenza.
#
# Recall status[] starts at E+I and counts down to REC=0.
#
# If I=7, E=2:
#   SUS REC                   T    E+I
#     |  |                    |     |
#    -1  0  1  2  3  4  5  6  7  8  9
#           |===================|

# Disease model. Each disease has a name, transmissivity coefficient,
# recovery coefficient, and exposure and infection times.
class Disease():
    def __init__(self, name='influenza', t=0.95, E=2, I=7, r=0.0):
        self.name=name
        self.t=t         # Transmissivity: how easy is it to pass on?
        self.E=E         # Length of exposure (in days)
        self.I=I         # Length of infection (in days)
        self.r=r         # Probability of lifelong immunity at recovery

# Agent model. Each agent has a susceptibility value, a vaccination
# state, and a counter that is used to model their current E, I, R or
# S status.
class Agent():
    def __init__(self, s=0.99):
        self.s = s       # Susceptibility: how frail is my immune system?
        self.v = 1.0     # Vaccination state
        self.c = -1      # Current state S=-1, R=0, E,I > 0
        self.disease = None

    # Return True if infectious (i.e., in I or E state), False
    # otherwise.
    def state(self):
        '''Returns True if agent is infectious.'''
        return(self.c > 0)

    # Set the agent's vaccination value to whatever value you give.
    def vaccinate(self, v):
        '''Models vaccination; v=0 denotes full immunity; v=1 denotes no immunity.'''
        self.v = v

    # Susceptible: if other is infected, roll the dice and update your
    # state. No real need to check other.state() here, since it is
    # checked prior to invoking the method, but it is included as per
```

```
    # spec. Note also that I add 1 to I+E, because my first step in
    # run() is to update state: your code may differ. Finally, it's
    # important to "remember" which disease you have so that you can
    # handle recovery and susceptibility correctly when the disease
    # finally runs its course.
    def infect(self, other, disease):
        '''Other tries to infects self with disease.'''
        if other.state() and self.c < 0 and rolldie(self.s*self.v*disease.t):
            self.c = disease.E + disease.I + 1
            self.disease = disease
            return(True)
        return(False)

    # Update the status of the agent. This involves decrementing your
    # internal counter if you are actively infected. When you get to
    # 0, you need to flip a (weighted) coin to decide if the agent
    # goes to state R (c=0) or back to state S (c=-1).
    def update(self):
        '''Daily status update.'''
        if self.c == 1:
            if not rolldie(self.disease.r):
                # Revert to susceptible, c=-1.
                self.c = -1
            else:
                # Lifelong immunity at recovery, c=0.
                self.c = 0
            # Clear your internal disease value.
            self.disease = None
        elif self.c > 1:
            # One day closer to recovery.
            self.c = self.c - 1
            return(True)
        return(False)

# Simulation model. Each simulation runs for at most a certain
# duration, D, expressed in terms of days.
class Simulation():
    def __init__(self, D=500):
        self.steps = D          # Maximum number of timesteps
        self.agents = None      # List of agents in the simulation
        self.disease = None     # Disease being simulated
        self.history = []       # History of (E, I) tuples
        self.m = 0.001          # Mixing parameter for this simulation

    # Populates the simulation with n agents and sets the mixing
    # parameter to m.
    def populate(self, n, m = 0.01):
        '''Populate simulation with n agents, having mixing probability m.'''
        self.m = m
        self.agents = []
        for i in range(n):
            self.join(Agent())

    # Add agent to current simulation.
    def join(self, agent):
        '''Add specified agent to current simulation.'''
        self.agents.append(agent)

    # Add disease to current simulation. For now, you can only model
    # one disease at a time.
    def introduce(self, disease):
        '''Add specified disease to current simulation.'''
        self.disease = disease
```

```
    # Seed the simulation with k agents having the specified disease.
    def seed(self, disease, k=1):
        '''Seed a certain number of agents with a particular disease.'''
        # Add the disease to the simulation.
        self.introduce(disease)
        # I+E+1, because my first step in run() is to update
        # state. Also, remember what disease you have.
        for agent in sample(self.agents, k):
            agent.c = disease.E + disease.I + 1
            agent.disease = disease

    # This is where the simulation actually happens. The run() method
    # performs at most self.steps iterations, where each iteration
    # updates the agents, counts how many are in E and I states,
    # checks if there is an early termination (i.e., no contagious
    # agents left) and then propagates the infection as per the mixing
    # parameter, m.
    def run(self):
        '''Run the simulation.'''
        for i in range(self.steps):
            # Update each agent, counting how many are still exposed
            # or infected.  Finding infected agents first avoids
            # letting the infection infect a friend's friend in one
            # pass.
            contagious = [ a for a in self.agents if a.update() ]
            # Update the history with exposed and infected counts.
            self.history.append((len([ a for a in contagious if a.c > self.disease.I ]),
                                 len([ a for a in contagious if a.c <= self.disease.I ])))
            # Exit early if there are no infected agents left.
            if self.history[-1] == (0, 0):
                return(i)
            for a1 in contagious:
                # Let's see who a1 can infect. No need to check
                # a2.state() here, as a2.infect() will check it for
                # you. Note the use of the mixing parameter to
                # determine if a1 and a2 have been in contact with
                # each other today.
                for a2 in self.agents:
                    if rolldie(self.m):
                        a2.infect(a1, self.disease)
        # Return the history of (E, I) tuples.
        return(self.history)

    # This method plots the pandemic curve from the self.history variable.
    def plot(self):
        '''Produce a pandemic curve for the simulation.'''
        plt.title('Simulation')
        plt.axis( [0, len(self.history), 0, len(self.agents)] )
        plt.xlabel('Days')
        plt.ylabel('N')
        plt.plot( [ i for i in range(len(self.history)) ], [ e for (e, i) in self.history ], '
g-', label='Exposed' )
        plt.plot( [ i for i in range(len(self.history)) ], [ i for (e, i) in self.history ], '
r-', label='Infected' )
        plt.show()

# A test function. Values to try:
#   test(500, 0.001, 1, 'influenza', 0.95, 2, 7, 0.8)
#   test(1000, 0.0005, 3, 'killmenow', 0.99, 3, 5, 1.0)
#   test(1000, 0.0005, 3, 'killmelater', 0.99, 4, 10, 0.6)
def test(n, m, k, name, t, E, I, r):
    disease = Disease(name, t, E, I, r)
```

```
    S = Simulation()
    S.populate(n, m)
    S.seed(disease, k)
    S.run()
    S.plot()
    return(S)

if __name__ == '__main__':
   test(10, 0.001, 1, 'influenza', 0.99, 2, 7, 0.8)
   test(500, 0.001, 1, 'influenza', 0.95, 2, 7, 0.8)
   test(1000, 0.0005, 3, 'killmenow', 0.99, 3, 5, 1.0)
   test(1000, 0.0005, 3, 'killmelater', 0.99, 4, 10, 0.6)
```