```
# DWAYNE FRASER
# PROBLEM 3
# Copyright 2017, 2013, 2011 Pearson Education, Inc., W.F. Punch & R.J.Enbody
"""Predator-Prey Simulation
   four classes are defined: animal, predator, prey, and island
   where island is where the simulation is taking place,
   i.e. where the predator and prey interact (live).
   A list of predators and prey are instantiated, and
   then their breeding, eating, and dying are simulted.
"""
import random
import pylab
import numpy as np

class Island (object):
    """Island
       n X n grid where zero value indicates not occupied."""
    def __init__(self, n, prey_count=0, predator_count=0, human_count = 0): # ADDED HUMAN
        '''Initialize grid to all 0's, then fill with animals
        '''
        # print(n,prey_count,predator_count)
        self.grid_size = n
        self.grid = []
        for i in range(n):
            row = [0]*n    # row is a list of n zeros
            self.grid.append(row)
        self.init_animals(prey_count,predator_count, human_count) # ADDED HUMAN

    def init_animals(self,prey_count, predator_count, human_count): # ADDED HUMAN
        ''' Put some initial animals on the island
        '''
        count = 0
        # while loop continues until prey_count unoccupied positions are found
        while count < prey_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_prey=Prey(island=self,x=x,y=y)
                count += 1
                self.register(new_prey)
        count = 0
        # same while loop but for predator_count
        while count < predator_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_predator=Predator(island=self,x=x,y=y)
                count += 1
                self.register(new_predator)

        count = 0
        # ADDED HUMAN LOOP
```

```
        while count < human_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_human = Human(island=self,x=x,y=y)
                count += 1
                self.register(new_human)

    def clear_all_moved_flags(self):
        ''' Animals have a moved flag to indicated they moved this turn.
        Clear that so we can do the next turn
        '''
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                if self.grid[x][y]:
                    self.grid[x][y].clear_moved_flag()

    def size(self):
        '''Return size of the island: one dimension.
        '''
        return self.grid_size

    def register(self,animal):
        '''Register animal with island, i.e. put it at the
        animal's coordinates
        '''
        x = animal.x
        y = animal.y
        self.grid[x][y] = animal

    def remove(self,animal):
        '''Remove animal from island.'''
        x = animal.x
        y = animal.y
        self.grid[x][y] = 0

    def animal(self,x,y):
        '''Return animal at location (x,y)'''
        if 0 <= x < self.grid_size and 0 <= y < self.grid_size:
            return self.grid[x][y]
        else:
            return -1  # outside island boundary

    def __str__(self):
        '''String representation for printing.
          (0,0) will be in the lower left corner.
        '''
        s = ""
        for j in range(self.grid_size-1,-1,-1):  # print row size-1 first
            for i in range(self.grid_size):      # each row starts at 0
                if not self.grid[i][j]:
                    # print a '.' for an empty space
```

```
                s+= "{:<2s}".format('.' + "  ")
            else:
                s+= "{:<2s}".format((str(self.grid[i][j])) + "  ")
        s+="\n"
    return s

def count_prey(self):
    ''' count all the prey on the island'''
    count = 0
    for x in range(self.grid_size):
        for y in range(self.grid_size):
            animal = self.animal(x,y)
            if animal:
                if isinstance(animal,Prey):
                    count+=1
    return count

def count_predators(self):
    ''' count all the predators on the island'''
    count = 0
    for x in range(self.grid_size):
        for y in range(self.grid_size):
            animal = self.animal(x,y)
            if animal:
                if isinstance(animal,Predator):
                    count+=1
    return count

def count_human(self): # ADDED HUMAN COUNT
    ''' count all the predators on the island'''
    count = 0
    for x in range(self.grid_size):
        for y in range(self.grid_size):
            animal = self.animal(x,y)
            if animal:
                if isinstance(animal, Human):
                    count+=1
    return count

class Animal(object):
    def __init__(self, island, x=0, y=0, s="A"):
        '''Initialize the animal's and their positions
        '''
        self.island = island
        self.name = s
        self.x = x
        self.y = y
        self.moved=False

    def position(self):
        '''Return coordinates of current position.
        '''
```

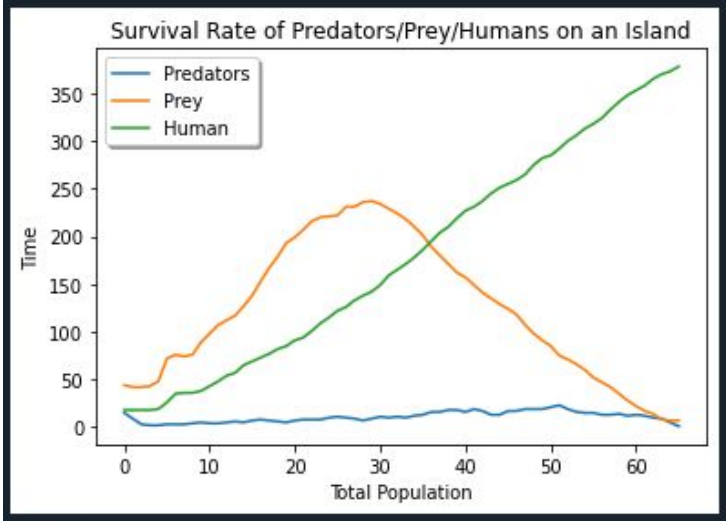
```
    return self.x, self.y

def __str__(self):
    return self.name

def check_grid(self,type_looking_for=int):
    ''' Look in the 8 directions from the animal's location
    and return the first location that presently has an object
    of the specified type. Return 0 if no such location exists
    '''
    # neighbor offsets
    offset = [(-1,1),(0,1),(1,1),(-1,0),(1,0),(-1,-1),(0,-1),(1,-1)]
    result = 0
    for i in range(len(offset)):
        x = self.x + offset[i][0]  # neighboring coordinates
        y = self.y + offset[i][1]
        if not 0 <= x < self.island.size() or \
           not 0 <= y < self.island.size():
            continue
        if type(self.island.animal(x,y))==type_looking_for:
            result=(x,y)
            break
    return result

def move(self):
    '''Move to an open, neighboring position '''
    if not self.moved:
        location = self.check_grid(int)
        if location:
            # print('Move, {}, from {},{} to {},{}'.format( \
            #       type(self),self.x,self.y,location[0],location[1]))
            self.island.remove(self)   # remove from current spot
            self.x = location[0]       # new coordinates
            self.y = location[1]
            self.island.register(self) # register new coordinates
            self.moved=True
def breed(self):
    ''' Breed a new Animal.If there is room in one of the 8 locations
    place the new Prey there. Otherwise you have to wait.
    '''
    if self.breed_clock <= 0:
        location = self.check_grid(int)
        if location:
            self.breed_clock = self.breed_time
            # print('Breeding Prey {},{}'.format(self.x,self.y))
            the_class = self.__class__
            new_animal = the_class(self.island,x=location[0],y=location[1])
            self.island.register(new_animal)

def clear_moved_flag(self):
    self.moved=False
```

```
class Prey(Animal):
    def __init__(self, island, x=0,y=0,s="O"):
        Animal.__init__(self,island,x,y,s)
        self.breed_clock = self.breed_time
        # print('Init Prey {},{}, breed:{}'.format(self.x, self.y,self.breed_clock))

    def clock_tick(self):
        '''Prey only updates its local breed clock
        '''
        self.breed_clock -= 1
        # print('Tick Prey {},{}, breed:{}'.format(self.x,self.y,self.breed_clock))

class Predator(Animal):
    def __init__(self, island, x=0,y=0,s="X"):
        Animal.__init__(self,island,x,y,s)
        self.starve_clock = self.starve_time
        self.breed_clock = self.breed_time
        # print('Init Predator {},{}, starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))

    def clock_tick(self):
        ''' Predator updates both breeding and starving
        '''
        self.breed_clock -= 1
        self.starve_clock -= 1
        # print('Tick, Predator at {},{} starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))
        if self.starve_clock <= 0:
            # print('Death, Predator at {},{}'.format(self.x,self.y))
            self.island.remove(self)

    def eat(self):
        ''' Predator looks for one of the 8 locations with Prey. If found
        moves to that location, updates the starve clock, removes the Prey
        '''
        if not self.moved:
            location = self.check_grid(Prey)
            if location:
                # print('Eating: pred at {},{}, prey at {},{}'.format( \
                #       self.x,self.y,location[0],location[1]))
                self.island.remove(self.island.animal(location[0],location[1]))
                self.island.remove(self)
                self.x=location[0]
                self.y=location[1]
                self.island.register(self)
                self.starve_clock=self.starve_time
                self.moved=True

class Human(Animal): # ADDED HUMAN CLASS

    def __init__(self, island, x = 0, y = 0, s = "H"): # Defines init for humans. This uses the starve clock, hunt time and
breed clocks
```

```
        Animal.__init__(self, island, x, y, s)# shows the 3 different attributes needed as x,y s
        self.starve_clock = self.starve_time # declares the starve clock will equal starve time
        self.hunt_time = self.hunt_time# declares the hunt time will equal hunt time
        self.breed_clock = self.breed_time# declares the breed clock will equal breed time

    def clock_tick(self): # Reused code from the Predator Class, clock_tick
        ''' Human updates both breeding and starving
        '''
        self.breed_clock -= 1 # allows the clock to incriment down
        self.starve_clock -= 1# allows the clock to incriment down
        if self.starve_clock <= 0: # if the starve clock is below 0, remove from island, they died
            self.island.remove(self)# remove from island

    def eat(self): # Define eat code, got this from above
        if not self.moved and self.hunt_time == 0:
            location = self.check_grid(Prey)
            if location:
                self.island.remove(self.island.animal(location[0], location[1]))
                self.island.remove(self)
                self.x = location[0]
                self.y = location[1]
                self.island.register(self)
                self.starve_clock = self.starve_time
                self.moved = True

##############################################
def main(predator_breed_time=6, predator_starve_time=3, initial_predators=15, prey_breed_time=5, initial_prey=52, \
         size=20, ticks=230, human_breed_time = 6, human_starve_time = 43, human_hunt_time = 13, inital_human =
18):
    ''' main simulation. Sets defaults, runs event loop, plots at the end
    '''
    # initialization values
    Predator.breed_time = predator_breed_time
    Predator.starve_time = predator_starve_time
    Prey.breed_time = prey_breed_time

    Human.breed_time = human_breed_time # ADDED HUMAN BREED
    Human.starve_time = human_starve_time # ADDED HUMAN STARVE
    Human.hunt_time = human_hunt_time # ADDED HUMAN HUNT

    # for graphing
    predator_list=[]
    prey_list=[]
    human_list = [] # # ADDED HUMAN

    # make an island
    isle = Island(size,initial_prey, initial_predators, inital_human) # ADDED HUMAN
    print(isle)# print the island

    # event loop.
    # For all the ticks, for every x,y location.
    # If there is an animal there, try eat, move, breed and clock_tick
```

```
    for i in range(ticks):
        # important to clear all the moved flags!
        isle.clear_all_moved_flags()
        for x in range(size):
            for y in range(size):
                animal = isle.animal(x,y)
                if animal:
                    if isinstance(animal,Predator) or isinstance(animal, Human): # ADDED HUMAN
                        animal.eat()
                    animal.move()
                    animal.breed()
                    animal.clock_tick()

        # record info for display, plotting
        prey_count = isle.count_prey()
        predator_count = isle.count_predators()
        human_count = isle.count_human() # ADDED HUMAN

        if prey_count == 0:
            print('Lost the Prey population. Quiting.')
            break
        if predator_count == 0:
            print('Lost the Predator population. Quitting.')
            break
        if human_count == 0: # ADDED HUMAN
            print("Lost the Human Population. Quitting.")
            break
        prey_list.append(prey_count)
        predator_list.append(predator_count)
        human_list.append(human_count) # ADDED HUMAN
        # print out every 10th cycle, see what's going on
        if not i%10:
            print(prey_count, predator_count, human_count)# added human count to this
        # print the island, hold at the end of each cycle to get a look
#       print('*'*20)
#       print(isle)
#       ans = input("Return to continue")
    pylab.plot(np.array(range(0,len(predator_list))), np.array(predator_list), label="Predators")
    pylab.plot(np.array(range(0,len(prey_list))), np.array(prey_list), label="Prey")
    pylab.plot(np.array(range(0,len(human_list))), np.array(human_list), label="Human") # ADDED HUMAN

    pylab.xlabel('Total Population')
    pylab.ylabel('Time')
    pylab.title('Survival Rate of Predators/Prey/Humans on an Island')

    pylab.legend(loc="best", shadow=True)
    pylab.show()

main()
```

```
In [1]: runfile('C:/Program Files (x86)/Work/Python/Python Dwayne Solutions/HW 3/p3_Fraser_Dwayne.py', wdir='C:/Program Files (x86)/Work/Python/Python
Dwayne Solutions/HW 3')
. . . . . . . . O . . . . O . O . . . O .
. . . . . . . . H . . . O . O . . . . . .
O . . . H . . . . . H O . . . . . . . . .
. . . . . . . . . . H O . . . . . . . . .
. . . . . . . . . H . . O . O . . O . . .
. . . X . . O . . . . . . . . . . . . . .
. . O H . X . H . . . . . O O . . . . . H
O . . . . . . H . . . . . O . . . . . . .
. . . . . O . . . X O . O . . H . . . . .
O . . . . . . . . . . . . . . . . O X . .
. . . . . . . H . . . O . . . . . . . X O
. . . X . . O O O H O . . O . . . . . . O
. O . . . . . O . . . . . O . . . . . . .
. . . . . . . . O . X . . . . . . . O X O
. . X . X . O . . . . . H . . O . . O . .
. . . . . O O . . . . . . . X . . . . H .
. . . . . . . . . . O . . . X . . H . . .
. O . . O . . . . O O . . H . . . . . . .
. . . . . . . . . . . X O . H . . . . O
. . O . X O . . . O . . . . . . X . . H

44 15 18
98 4 43
199 7 91
234 11 149
157 16 227
85 21 285
22 13 353
Lost the Predator population. Quitting.
```




```
In [2]:
```