Python Programming

Homework 1

Jordan Diaz

Solution #1:

# This program experiments with classes in python

class NVector(object):  
 def \_\_init\_\_(self, \*sequence):  
 *""" Constructor, creates a list of elements in the sequence"""* self.elements\_of\_sequence = []  
 if len(sequence) == 1:  
 for iterable in sequence:  
 for index in iterable:  
 self.elements\_of\_sequence.append(index)  
 elif len(sequence) > 1:  
 for element in sequence:  
 self.elements\_of\_sequence.append(element)  
  
 def \_\_len\_\_(self):  
 *""" Returns the length of the list from the elements in the sequence"""* return len(self.elements\_of\_sequence)  
  
 def \_\_getitem\_\_(self, index):  
 *""" Returns the index from the list of elements in the sequence"""* return self.elements\_of\_sequence[index]  
  
 def \_\_setitem\_\_(self, index, value):  
 *""" Modifies the value at an index from the list of elements in the sequence"""* self.elements\_of\_sequence[index] = value  
  
 def \_\_str\_\_(self):  
 return str(self.elements\_of\_sequence)  
  
 def \_\_eq\_\_(self, other):  
 if type(other) == NVector:  
 if self.elements\_of\_sequence == other.elements\_of\_sequence:  
 return True  
 else:  
 return False  
 else:  
 return False  
  
 def \_\_ne\_\_(self, other):  
 if type(other) == NVector:  
 if self.elements\_of\_sequence != other.elements\_of\_sequence:  
 return True  
 else:  
 return False  
 else:  
 return True  
  
 def \_\_add\_\_(self, other):  
 *""" Adds NVector or number if it is the right of it"""* temp\_list = []  
 if type(other) == NVector:  
 list\_a = []  
 list\_b = []  
  
 for items in self.elements\_of\_sequence:  
 list\_a.append(items)  
 for items in other.elements\_of\_sequence:  
 list\_b.append(items)  
  
 if len(list\_a) > len(list\_b):  
 for i in range(0, abs(len(list\_a) - len(list\_b))):  
 list\_b.append(0)  
 elif len(list\_a) < len(list\_b):  
 for j in range(0, abs(len(list\_a) - len(list\_b))):  
 list\_a.append(0)  
  
 for k in range(0, len(list\_a)):  
 temp\_list.append(list\_a[k] + list\_b[k])  
  
 elif type(other) == int or type(other) == float:  
 for element in self.elements\_of\_sequence:  
 temp\_list.append(other + element)  
  
 return NVector(temp\_list)  
  
 def \_\_radd\_\_(self, other):  
 *""" Adds a number if it is the the Left of it to each element of the sequence, must be number"""* temp\_list = []  
 if type(other) == int or type(other) == float:  
 for element in self.elements\_of\_sequence:  
 temp\_list.append(other + element)  
 return NVector(temp\_list)  
  
 def \_\_mul\_\_(self, other):  
 *""" multiplies NVector of numbers or number if it is the right of it, NVectors must be same size"""* temp\_value = 0  
 if type(other) == NVector:  
 for i in range(0, len(self.elements\_of\_sequence)):  
 temp\_value += self.elements\_of\_sequence[i] \* other[i]  
 elif type(other) == int or type(other) == float:  
 for j in range(0, len(self.elements\_of\_sequence)):  
 temp\_value += self.elements\_of\_sequence[j] \* other  
 return temp\_value  
  
 def \_\_rmul\_\_(self, other):  
 *""" multiplies a number if it is the the Left of it to each element of the sequence, must be number"""* temp\_value = 0  
 if type(other) == int or type(other) == float:  
 for j in range(0, len(self.elements\_of\_sequence)):  
 temp\_value += self.elements\_of\_sequence[j] \* other  
 return temp\_value  
  
 def zeros(self, n):  
 *""" returns a new NVector object with dimension n with all elements 0"""* temp\_list = []  
 for i in range(0, n):  
 temp\_list.append(0)  
 return NVector(temp\_list)  
  
  
def testif(b, testname, msgOK="", msgFailed=""):  
 *""" Unit Testing"""* if b:  
 print("Success: " + testname + "; " + msgOK)  
 else:  
 print("Failed: " + testname + "; " + msgFailed)  
 return b  
  
  
def main():  
 *""" used for testing of the NVector class"""* # testing part \_\_init\_\_, \_\_eq\_\_, \_\_ne\_\_, \_\_setitem\_\_  
 a1 = NVector([3, 0, 1, -1])  
 a2 = NVector(3, 0, 1, -1)  
 testif(a1 == a2, "\_\_init\_\_, \_\_eq\_\_", "constructor works and eq works", "constructor works or eq failed")  
 a2[1] = 10  
 testif(a2[1] == 10 and a1 != a2, "\_\_setitem\_\_, \_\_ne\_\_", "setitem works and ne works", "setitem or ne failed")  
  
 # testing part \_\_len\_\_, \_\_getitem\_\_  
 testif(a1.\_\_len\_\_() == 4 and a1.\_\_getitem\_\_(0) == 3, "\_\_len\_\_, \_\_getitem\_\_", "these worked", "these failed")  
  
 # testing \_\_str\_\_  
 testif(type(a1.\_\_str\_\_()) == str, "\_\_str\_\_", "worked", "failed")  
  
 # testing \_\_add\_\_ , \_\_radd\_\_  
 b1 = NVector(3, 0, 1, -1)  
 b2 = NVector(1, 2, 3, 4)  
 expected = [4, 2, 4, 3]  
 testif((b1 + b2).elements\_of\_sequence == expected and b1 + 10 == 10 + b1, "\_\_add\_\_, \_\_radd\_\_", "worked", "failed")  
  
 # testing \_\_mul\_\_ and \_\_rmul\_\_  
 testif(b1 \* b2 == 2 and b1 \* 10 == 30 and 10 \* b1 == 30, "\_\_mul\_\_, \_\_rmul\_\_", "worked", "failed")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal for problem 1:

Text

Description automatically generated

Solution 2:

# this program experiments with class inheritance  
class Product(object):  
 def \_\_init\_\_(self, name, mass=0.0, stock=0, price=0.0):  
 *""" Constructor"""* self.product\_name = name  
 self.product\_mass = float(mass)  
 self.product\_stock = stock  
 self.product\_price = float(price)  
  
 def \_\_str\_\_(self):  
 return "{}, ${}, {} kg, {} in stock".format(self.product\_name, self.product\_price, self.product\_mass,  
 self.product\_stock)  
  
 def name(self):  
 return self.product\_name  
  
 def mass(self):  
 return self.product\_mass  
  
 def stock(self):  
 return self.product\_stock  
  
 def price(self):  
 return self.product\_price  
  
 def set\_price(self, new\_price):  
 self.product\_price = float(new\_price)  
  
  
class DiscountedProduct(Product):  
 def \_\_init\_\_(self, discount, product):  
 Product.\_\_init\_\_(self, product.product\_name, product.product\_mass, product.product\_stock,  
 product.product\_price - (discount \* product.product\_price))  
 self.product\_discount = discount  
 self.base\_product = product  
  
 def \_\_str\_\_(self):  
 *""" prints out the data from the discounted product"""* # refreshes the data  
 self.\_\_init\_\_(self.product\_discount, self.base\_product)  
 return "discounted {}%: {}, ${}, {} kg, {} in stock".format(self.product\_discount \* 100, self.product\_name,  
 self.product\_price, self.product\_mass,  
 self.product\_stock)  
  
  
def main():  
 *""" Main function"""* p = Product(name="Lava lamp", price=30, mass=0.8, stock=123)  
 disc\_p = DiscountedProduct(0.2, p)  
 print(p)  
 print(disc\_p)  
 p.set\_price(20)  
 print(p.price())  
 print(disc\_p)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal session for problem 2

Text

Description automatically generated

Solution #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 time  
import pylab  
  
  
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):  
 *"""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)  
  
 def init\_animals(self, prey\_count, predator\_count, human\_count):  
 *""" 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  
 # same while loop but for human\_count  
 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\_humans(self):  
 *""" count all the humans 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):  
 def \_\_init\_\_(self, island, x=0, y=0, s="H"):  
 Animal.\_\_init\_\_(self, island, x, y, s)  
 self.starve\_clock = self.starve\_time  
 self.breed\_clock = self.breed\_time  
 self.hunt\_clock = self.hunt\_time  
  
 def clock\_tick(self):  
 *""" Predator updates both breeding and starving  
 """* self.breed\_clock -= 1  
 self.starve\_clock -= 1  
 self.hunt\_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:  
 if self.hunt\_clock <= 0:  
 location = self.check\_grid(Predator)  
 if location:  
 #print('Eating: human at {},{}, predator 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.hunt\_clock = self.hunt\_time  
 self.moved = True  
  
  
###########################################  
def main(predator\_breed\_time=4, predator\_starve\_time=3, initial\_predators=35, prey\_breed\_time=1, initial\_prey=59,  
 human\_breed\_time=8, human\_starve\_time=13, human\_hunt\_time=12, initial\_humans=3, size=12, ticks=1000):  
 *""" 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  
 Human.starve\_time = human\_starve\_time  
 Human.hunt\_time = human\_hunt\_time  
  
 # for graphing  
 predator\_list = []  
 prey\_list = []  
 human\_list = []  
  
 # make an island  
 isle = Island(size, initial\_prey, initial\_predators, initial\_humans)  
 print(isle)  
  
 # 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):  
 animal.eat()  
 if isinstance(animal, 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\_humans()  
  
 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:  
 print('Lost the Human population. Quitting.')  
 break  
  
 prey\_list.append(prey\_count)  
 predator\_list.append(predator\_count)  
 human\_list.append(human\_count)  
  
 # print out every 10th cycle, see what's going on  
 if not i % 10:  
 print("prey: {}, predator: {}, human: {}".format(prey\_count, predator\_count, human\_count))  
 # 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(range(0, ticks), predator\_list, label="Predators")  
 pylab.plot(range(0, ticks), prey\_list, label="Prey")  
 pylab.plot(range(0, ticks), human\_list, label="Human")  
 pylab.legend(loc="best", shadow=True)  
 pylab.show()  
 print(isle)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Graph for problem 1

Chart, histogram

Description automatically generated

Terminal Session for problem #1:

First tick

A picture containing keyboard, electronics

Description automatically generated

Last tick

A picture containing electronics, keyboard

Description automatically generated