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:

```
Success: __init__, __eq__; constructor works and eq works
Success: __setitem__, __ne__; setitem works and ne works
Success: __len__, __getitem__; these worked
Success: __str__; worked
Success: __add__, __radd__; worked
Success: __mul__, __rmul__; worked
```

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

```
Lava lamp, $30.0, 0.8 kg, 123 in stock
discounted 20.0%: Lava lamp, $24.0, 0.8 kg, 123 in stock
20.0
discounted 20.0%: Lava lamp, $16.0, 0.8 kg, 123 in stock
```

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, -
```

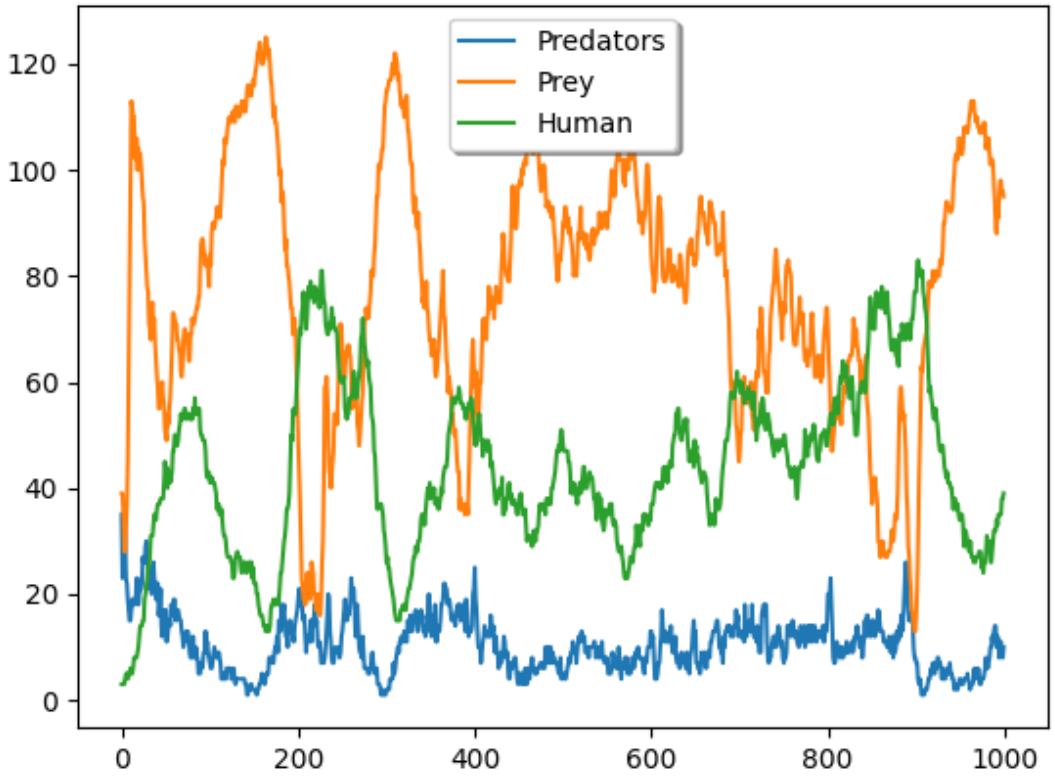
```
1), (1, -1)]
        result = 0
        for i in range(len(offset)):
            x = self.x + offset[i][0]  # neighboring coordinates
            y = self.y + offset[i][1]
            if not 0 <= x < self.island.size() or \
                    not 0 <= y < self.island.size():
                continue
            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

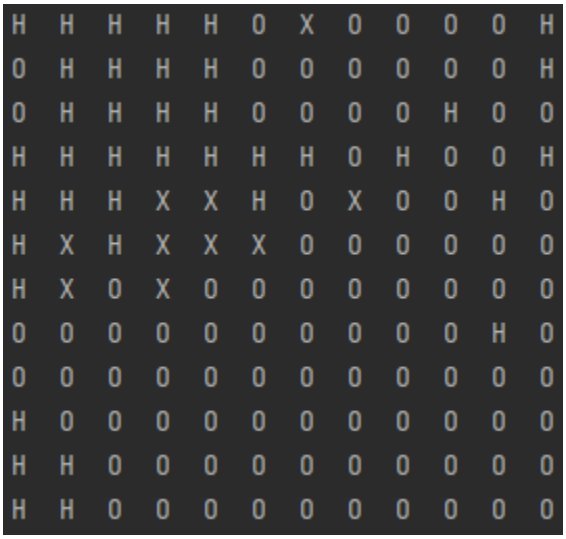


Terminal Session for problem #1:

First tick

```
O  O  .  O  O  O  .  X  O  X  X  O
O  .  O  O  O  O  O  O  O  .  X  X
X  O  O  O  O  O  .  .  O  O  .  O
.  O  .  .  O  O  X  O  X  O  X  O
O  O  X  .  .  X  H  O  .  .  O  .
X  O  O  X  O  .  O  O  H  .  X  H
.  .  O  .  X  .  X  .  X  .  .  X
.  X  X  O  O  X  X  X  .  .  .  O
X  .  O  O  O  .  .  .  .  O  O  .
O  O  O  .  X  X  X  X  O  O  O  .
.  X  X  X  O  .  .  .  .  O  .  O
X  X  .  O  .  X  .  .  O  O  .  X
```

Last tick


H H H H H O X O O O O H
O H H H H O O O O O O H
O H H H H O O O O H O O
H H H H H H H O H O O H
H H H X X H O X O O H O
H X H X X X O O O O O O
H X O X O O O O O O O O
O O O O O O O O O O H O
O O O O O O O O O O O O
H O O O O O O O O O O O
H H O O O O O O O O O O
H H O O O O O O O O O O