Rylee Texter

**Problem One**

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 14 16:02:33 2024

@author: rylee
"""

class Mstring():

    #initializes an instance of Mstring and sets the string attribute
    def __init__(self, obj):
        self.__string = str(obj)

    #returns the length of the current Mstring
    def __len__(self):
        return len(self.__string)

    #returns a string of the Mstring
    def __str__(self):
        return (self.__string)

    #returns a representation of the Mstring using __str__
    def __repr__(self):
        return(self.__str__())

    #adds an Mstring to another Mstring
    def __add__(self, mstring):
        new_string = self.__string + mstring.__string
        mstring = Mstring(new_string)
        return (mstring)

    #adds a string to an Mstring
    def __radd__(self, string):
        new_string = self.__string + string
        mstring = Mstring(new_string)
        return (mstring)

    #returns an item at a given index
    def __getitem__(self, index):
        try:
```

```
            return self.__string[index]
        except IndexError:
            return("index out of range")

    #sets a given character at a given index
    def __setitem__(self, index, char):
        try:
            chars = []

            ##loops through self.__string and adds them to a list chars
            for cha in self.__string:
                chars.append(cha)

            #changes the char at given index
            chars[index] = char
            mstring = Mstring('')

            #loops through chars list and uses __radd__ to create a
            #mstring with the correct string after looping
            for cha in chars:
                mstring = mstring.__radd__(cha)

            #sets the self.__string to the mstring.__string (string with change)
            self.__string = mstring.__string
        except IndexError as e:
            return("Please enter a valid index")

    #returns if two Mstrings are equal
    def __eq__(self, mstring2):
        if (isinstance(mstring2, str)):
            return(self.__string == mstring2)
        elif isinstance(mstring2, Mstring):
            return(self.__string == mstring2.__string)
        else:
            return(False)

    #returns if Mstrings do not equal eachother
    def __ne__(self, mstring2):
        if self.__eq__(mstring2):
            return False
        else:
            return True

    #replaces a substring in an Mstring with another substring
```

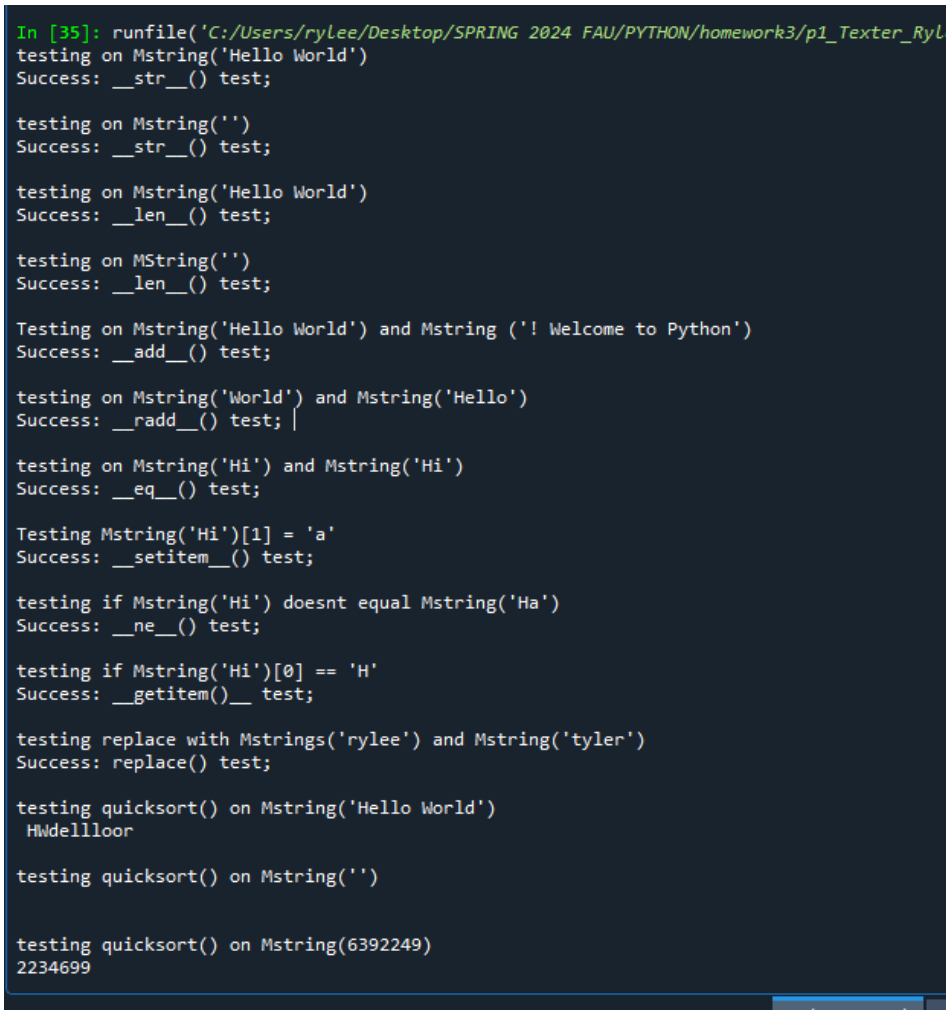
```
    #returns index of substring or -1 if not found
    def replace(self, s, t):
        if s in self.__string:

            #loops through characters in the self Mstring
            #to find index of substring
            for i in range(len(self.__string)):

                #checks if the current index is the substring
                if self.__string[i:i+len(s)] == s:
                    print('here')
                    rang = range(i, i+len(s))
                    count = 0

                    #loops through the substring length and sets the
                    #characters with the new substring
                    for j in rang:
                        self.__setitem__(t[count], j)
                        count += 1
                    return i
        else:
            return (-1)

    #finds the index of a substring in the Mstring and returns the
    #index
    def find(self, substring):
        if substring in self.__string:
            index = self.replace(substring, substring)
            return index
        else:
            return -1

#function to test methods and their success/failure
def testif(b, testname, msgOK='', msgFailed=''):
        if b:
            print("Success: " + testname+ "; "+msgOK)
        else:
            print("Failure:"+testname+";"+msgFailed)
        return b


##tests each Mstring method
def unit_tests():
    #testing __str__
```

```
print("testing on Mstring('Hello World')")
ms1 = Mstring("Hello World")
testif(str(ms1) == "Hello World", "__str__() test")
print()

print("testing on Mstring('')")
em1 = Mstring('')
testif(str(em1) == '', "__str__() test")
print()

#testing __len__
print("testing on Mstring('Hello World')")
testif(len(ms1)==11, "__len__() test")
print()

print("testing on MString('')")
testif(len(em1)==0, "__len__() test")
print()

#testing __add__
print("Testing on Mstring('Hello World') and Mstring ('! Welcome to Python')")
ms2 = Mstring("! Welcome to Python")
ms3 = ms1+ms2
testif(ms3 == "Hello World! Welcome to Python", "__add__() test")
print()

#testing __radd__
print("testing on Mstring('World') and Mstring('Hello')")
string = "World"
ms4 = Mstring("Hello")
ms5 = string + ms4
testif(ms5 == "HelloWorld", "__radd__() test")
print()


#testing __eq__
print("testing on Mstring('Hi') and Mstring('Hi')")
ms6 = Mstring("Hi")
ms7 = Mstring("Hi")
testif(ms6 == ms7, "__eq__() test")
print()

#testing __setitem__
print("Testing Mstring('Hi')[1] = 'a'")
```

```
    ms6[1] = 'a'
    testif(ms6 == "Ha", "__setitem__() test")
    print()

    #testing __ne__
    print("testing if Mstring('Hi') doesnt equal Mstring('Ha')")
    testif(ms6 != ms7, "__ne__() test")
    print()

    #testing __getitem__
    print("testing if Mstring('Hi')[0] == 'H'")
    char = ms7[0]
    testif(char == "H", "__getitem()__ test")
    print()


    #testing replace
    print("testing replace with Mstrings('rylee') and Mstring('tyler')")
    ms8 = "Hello my name is rylee"
    ms8 = (ms8.replace("rylee", "tyler"))
    testif(ms8 == "Hello my name is tyler", "replace() test")
    print()



##returns the sorted string of the Mstring
def quicksort(mstring):
    string = str(mstring)
    sorted_string = ''.join(sorted(string))
    return(sorted_string)


#tests the quicksort functioin on different Mstrings
def test_sort():
    print("testing quicksort() on Mstring('Hello World')")
    ms1 = Mstring("Hello World")
    print(quicksort(ms1))
    print()

    print("testing quicksort() on Mstring('')")
    ms2 = Mstring("")
    print(quicksort(ms2))
    print()
```

```
    print("testing quicksort() on Mstring(6392249)")
    ms3 = Mstring(6392249)
    print(quicksort(ms3))
    print()


#tests unit_tests() function and test_sort() function
def main():
    unit_tests()
    test_sort()

main()
```

**Output Screenshot:**



**Problem Two**

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 16 14:36:26 2024

@author: rylee
"""

# 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  # replaced by:
import matplotlib.pyplot as plt


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
        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 = 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

    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




##############################################
def main(predator_breed_time=6, predator_starve_time=4, initial_predators=8,
prey_breed_time=2, initial_prey=60, \
        human_starve_time = 5, human_breed_time = 4, initial_humans = 8, size=10,
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

# 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()
                elif 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(i)
        print('Lost the Prey population. Quiting.')
        break
    if predator_count == 0:
        print(i)
```

```
            print('Lost the Predator population. Quitting.')
            break
        if human_count == 0:
            print(i)
            print("Lost the Human population, Quitting")
            break
        human_list.append(human_count)
        prey_list.append(prey_count)
        predator_list.append(predator_count)
        # print out every 10th cycle, see what's going on
        if not i%10:
            print(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")
    plt.plot(predator_list)
    plt.plot(prey_list)
    plt.plot(human_list)
    plt.show()
    print(isle)

main()
```

**Output Screenshots:**

```
In [92]: runfile('C:/Users/rylee/Desktop/SPRING 2024 FAU/PYTHON/homew
2024 FAU/PYTHON/homework3')
.  O  .  .  X  .  O  O  O  H
.  O  .  O  O  .  O  O  O  O
O  X  O  O  O  O  .  X  O  .
O  H  O  O  .  O  H  O  O  H
.  .  O  O  O  .  O  .  X  H
.  O  X  O  O  O  O  .  O  O
O  .  O  .  O  X  O  X  O  O
.  O  .  O  .  H  .  O  O  O
O  O  H  O  O  O  O  O  O  O
O  O  .  O  .  X  O  H  O  O

45 8 8
34 8 10
61 10 9
58 15 8
58 16 9
59 13 6
53 9 11
63 9 9
52 15 9
60 15 8
47 5 11
61 4 18
53 8 12
53 6 18
57 2 21
58 5 15
60 8 13
57 9 14
52 5 13
60 6 13
54 4 17
59 5 14
62 5 20
43 8 16
62 4 19
248
Lost the Predator population. Quitting.
H  .  .  H  H  H  H  .  .  .
H  .  .  .  .  .  .  .  .  .
H              H  H  H  H
```

IPython Cons

```
58 5 15
60 8 13
57 9 14
52 5 13
60 6 13
54 4 17
59 5 14
62 5 20
43 8 16
62 4 19
248
Lost the Predator population. Quitting.
H  .  .  H  H  H  H  .  .  .
H  .  .  .  .  .  .  .  .  .
H  .  .  .  .  H  H  H  H  .
.  O  .  H  .  H  O  O  .  .
H  H  H  H  H  O  O  O  O  .
H  O  H  O  O  O  O  O  O  .
O  O  O  O  O  O  O  O  O  .
O  O  O  O  O  O  O  O  O  .
O  O  O  O  O  O  O  O  O  .
O  O  O  O  O  O  O  O  O  O


In [93]:
```