Homework 5

Jamie Andrews

Problem 1:

```
# -----------------Homework 5 Problem 1 by Jamie Andrews----------------
#1. Create a new class for Human deriving from the class Animal.
#2. Create a def function called __init__ and let that be the constructor of the class.
#3. Create another def function and call it clock_tick.
#4. Create another def function and call it eat.
#5. Create another def function and call it hunt.
#6. In superclass island, create a def function called count_human
#7. In superclass island, edit a def function called init_animals
#8. In superclass island, edit the constructor.
#9. Edit the main def funtion.
#10. Edit the main code.
"""Predator-Prey Simulation
   five classes are defined: animal, predator, prey, human 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."""
    #8.
    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)

#7.
    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 loop continues until prey_count unoccupied positions are found
        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
```

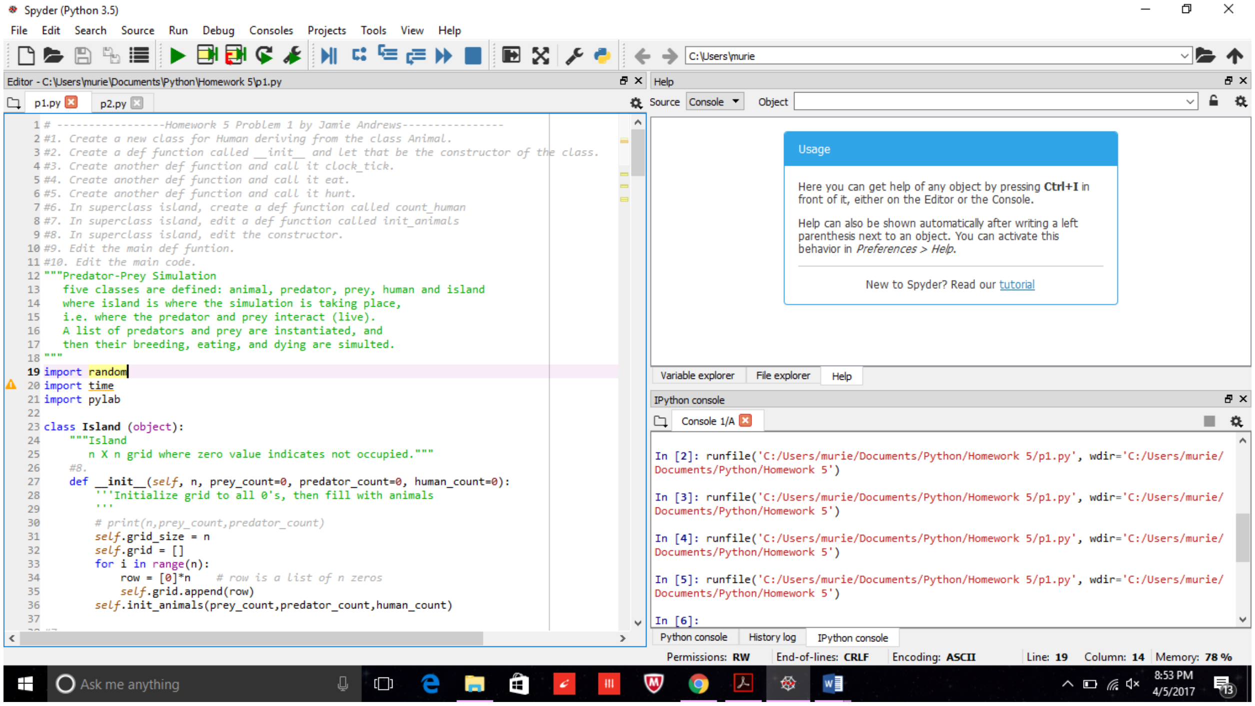
```
#6.
    def count_human(self):
        ''' count all the humans on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x,y)
                if animal:
                    if isinstance(animal,Human):
                        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
```

```python
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


#1.
class Human(Animal):
    #2.
    def __init__(self, island, x=0,y=0,s="H"):
        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))
     #3.
    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))

     #4.
    def eat(self):
        ''' Human 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: Human 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

    #5.
    def hunt(self):
        '''Human hunts for one of the 8 locations with Predator for fun or for
        sport. If found moves to that location, updates the starve clock, removes the
        Predator
        '''
        if not self.moved:
            location = self.check_grid(Predator)
            if location:
                # print('Hunting: Human at {},{}, pred 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


    ###############################################
    #9.
    def main(predator_breed_time=6, predator_starve_time=3, initial_predators=10, prey_breed_time=3, initial_prey=50, \
          size=10, ticks=300, initial_humans = 10, human_breed_time=4):
        ''' 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

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

        # make an island
        isle = Island(size,initial_prey, initial_predators,initial_humans)
        print(isle)

        # event loop.
```

```python
# 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()
                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()
    if prey_count == 0:
        print('Lost the Prey population. Quiting.')
        break
    if predator_count == 0:
        print('Lost the Predator 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_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")
#10.
    pylab.plot(predator_list, label="Predators")
    pylab.plot(prey_list, label="Prey")
    pylab.plot(human_list, label='Human')
    pylab.legend(loc="best", shadow=True)
    pylab.show()
```


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\murie
Editor - C:\Users\murie\Documents\Python\Homework 5\p1.py
p1.py
p2.py
Help
Source Console
Object
Variable explorer File explorer Help
IPython console
Console 1/A
Python console History log IPython console
Ask me anything
8:53 PM
4/5/2017

```
# ----------------Homework 5 Problem 1 by Jamie Andrews---------------
#1. Create a new class for Human deriving from the class Animal.
#2. Create a def function called __init__ and let that be the constructor of the class.
#3. Create another def function and call it clock_tick.
#4. Create another def function and call it eat.
#5. Create another def function and call it hunt.
#6. In superclass island, create a def function called count_human
#7. In superclass island, edit a def function called init_animals
#8. In superclass island, edit the constructor.
#9. Edit the main def funtion.
#10. Edit the main code.
"""Predator-Prey Simulation
   five classes are defined: animal, predator, prey, human 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."""
    #8.
    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)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [2]: runfile('C:/Users/murie/Documents/Python/Homework 5/p1.py', wdir='C:/Users/murie/
Documents/Python/Homework 5')

In [3]: runfile('C:/Users/murie/Documents/Python/Homework 5/p1.py', wdir='C:/Users/murie/
Documents/Python/Homework 5')

In [4]: runfile('C:/Users/murie/Documents/Python/Homework 5/p1.py', wdir='C:/Users/murie/
Documents/Python/Homework 5')

In [5]: runfile('C:/Users/murie/Documents/Python/Homework 5/p1.py', wdir='C:/Users/murie/
Documents/Python/Homework 5')

In [6]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **ASCII** Line: **19** Column: **14** Memory: **78 %**

Problem 2:

```
#----------------Homework 5 Problem 2 by Jamie Andrews---------------
#1. Create a def function called ed_read which returns the file as a string
#2. Create a def function called ed_find which searches the string through filename and returns a list.
#3. Create a def function called ed_replace which replaces search_str in the file with string replace_with.
```

#4. Create a def function called ed_append which appends the string to the end of the file.

#5. Create a def function called ed_write which writes to the file as position pos the string s.

#6. Create a def funtion called ed_insert which inserts into the file content.

```
#4.
def ed_append(filename,string):
    pass


#1.
def ed_read(filename, x1 = 0, x2 = -1):
    with open(filename, 'r') as f:
        for i, j in enumerate(f):
            if x1<=i<=x2:
                print(j)
            elif i>x2:
                break
#2.
def ed_find(filename, search_str):
    pass


#3.
def ed_replace(filename, search_str, replace_with, occurence = -1):
    pass
 #5.
def ed_write(filename, pos_str_col):
    pass
 #6.
def ed_insert(fn):
    pass

fn = "file1.txt" # assume this file does not exist yet.
ed_append(fn, "0123456789") # this will create a new file
ed_append(fn, "0123456789") # the file content is: 01234567890123456789
print(ed_read(fn, 3, 9)) # prints 345678. Notice that the interval excludes index to (9)
print(ed_read(fn, 3)) # prints from 3 to the end of the file: 34567890123456789
lst = ed_find(fn, "345")
print(lst) # prints [3, 13]
print(ed_find(fn, "356")) # prints []
ed_replace(fn, "345", "ABCDE", 1) # changes the file to 0123456789012ABCDE6789
# assume we reset the file content to 01234567890123456789 (not shown)
ed_replace(fn, "345", "ABCDE") # changes the file to 012ABCDE6789012ABCDE6789
# assume we reset the file content to 01234567890123456789 (not shown)
# this function overwrites original content:
ed_write(fn, ((2, "ABC"), (10, "DEFG"))) # changes file to: 01ABC56789DEFG456789
# this should work with lists as well: [(2, "ABC"), (10, "DEFG")]
# assume we reset the file content to 01234567890123456789 (not shown)
ed_write(fn, ((2, "ABC"), (30, "DEFG"))) # fails. raises ValueError("invalid position 30")
# assume we reset the file content to 01234567890123456789 (not shown)
# this function inserts new text, without overwriting:
ed_insert(fn, ((2, "ABC"), (10, "DEFG")))
# changed file to: 01ABC23456789DEFG0123456789
```