Kevin Leary

Homework 4

**Problem 1**

# 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

#loop for Humans

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(Predator):

def \_\_init\_\_(self, island, x=0, y=0, s="H"):

Predator.\_\_init\_\_(self,island,x,y,s)

self.hunt\_clock = self.hunt\_time

def clock\_tick(self):

''' Human updates hunt cloak and both predator clocks

'''

self.hunt\_clock -=1

Predator.clock\_tick(self)

def hunt(self):

''' Human looks for Predators to hunt for fun. NOT TO EAT

'''

if not self.moved:

location = self.check\_grid(Predator)

#print(location)

if location == (1,0) or location == (0,1) or location == (1,1):

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.hunt\_clock = self.hunt\_time

self.moved=True

###########################################

def main(human\_breed\_time=10, human\_starve\_time=7, human\_hunt\_time=5, initial\_humans=5,\

predator\_breed\_time=6, predator\_starve\_time=3, initial\_predators=10, prey\_breed\_time=3, initial\_prey=60, \

size=10, ticks=300):

''' 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 Initialization

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.hunt()

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

#NOW only quit if prey population is gone

#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")

pylab.plot(predator\_list)

pylab.plot(prey\_list)

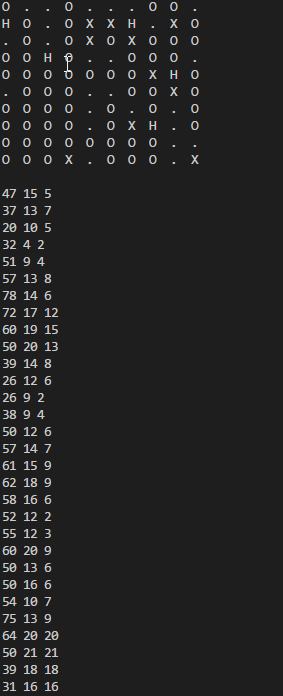
pylab.plot(human\_list)

pylab.gca().legend(('Predators', 'Prey', 'Humans'))

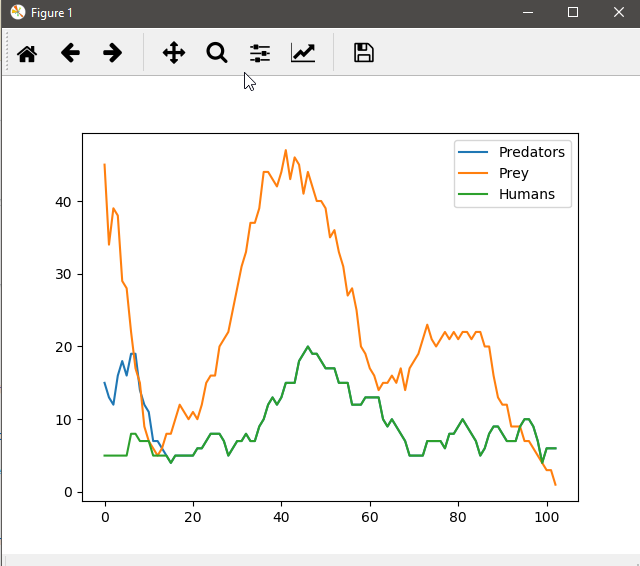
pylab.show()

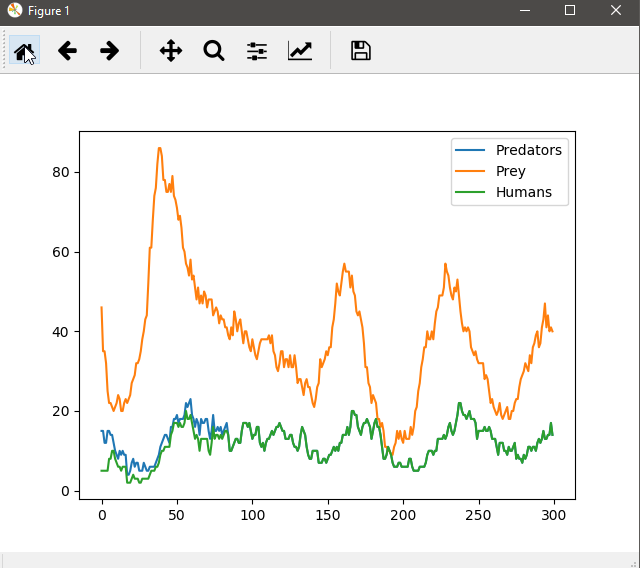
main()

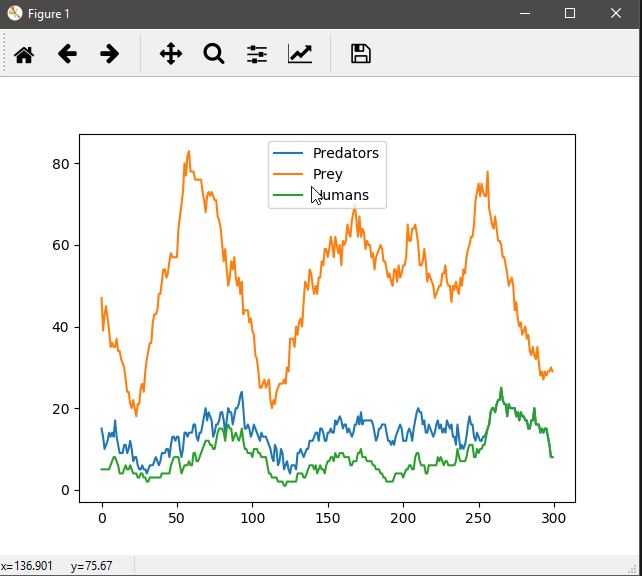
Terminal output for the last graph



Screenshots of different outcomes







**Problem 2**

#Part B

def test\_ed\_append():

test\_name = 'test\_append'

test\_fn = 'test.txt'

if os.path.isfile(test\_fn):

os.remove(test\_fn)

initial\_text = 'ABCD'

with open(test\_fn, 'w') as test\_f:

test\_f.write(initial\_text)

try:

new\_text = '01234'

ret = ed\_append(test\_fn, new\_text)

expected\_text = initial\_text + new\_text

current\_text = open(test\_fn, 'r').read()

cond = (ret == len(new\_text)) and (current\_text == expected\_text)

return testif(cond, test\_name)

except Exception as exc:

print('test {} failed due to exception: {}\n'.format(test\_name, str(exc)))

return False

def testif(b, testname, msgOK= "" , msgFailed= "" ):

"""Function used for testing."""

if b:

print ( "Success: " + testname + "; " + msgOK)

else :

print ( "Failed: " + testname + "; " + msgFailed)

return b

#Part A

def ed\_read(file, start, end):

"""Will Read the file starting from the give starting index and

ending at the given end index

"""

try:

str\_read = ""

str\_file = open(file, "r", encoding='utf8')

for start, c in str\_file:

if start == end:

break

else:

str\_read = str\_read + c

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

except ValueError:

print('one or more variables exceed file length')

break

def ed\_find(file, search\_str):

"""This function will retun the first index a word is found and can return

multiple values. will return [] if nothing is found

"""

try:

temp = list() #for variables

str\_file = open(file, "r", encoding='utf8')

for i, c in str\_file: #start searching the file

if c == search\_str[0]: #if the character in the file is equal to the first letter in the search string then...

temp.append(i) #append it to list then..

for j in search\_str: #incrementing through the search word

if str\_file[i+j] != search\_str[j]: #if it doesnt match the part of the file then....

temp.remove(i) #Remove the value from the list and end the loop

break

return temp

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

def ed\_replace(file, search\_str, replace\_with, occurence):

"""Similar to ed\_find .. can probably call it...

This function takes a search string and a astring to replace with..

if occurence >= 0 it replaces the search string with the replace word only at that index

if occurence == -1 then it replaces all of them

"""

try:

str\_file = open(file, "w", encoding='utf8')

if occurence == -1:

temp\_ls = ed\_find(file, search\_str)

for i in temp\_ls:

#temp\_str = ""

#for j in search\_str:

#temp\_str = str\_file[temp\_ls + j]

str\_file[i].replace(search\_str, replace\_with) #hoping that this works like a string

else:

str\_file[occurence].replace(search\_str, replace\_with)

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

def ed\_append(file, str\_a):

"""Appends a string to the end of the file. If the file does not exist a new one is created

The function returns the number of characters written to the file

"""

try:

str\_file = open(file, "a", encoding='utf8')

str\_file.write(str\_a)

return len(str\_a)

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

def ed\_write(file, pos\_str\_col):

"""pos\_str\_col is a list has (position, s)

"""

try:

str\_file = open(file, "r+", encoding='utf8')

count = 0

for i in pos\_str\_col:

if i < 0 or i > len(str\_file):

raise ValueError('position parameter is less than 0 or greater than the file length')

return count

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

def ed\_insert(file, pos\_str\_col):

"""

"""

try:

str\_file = open(file, "r+", encoding='utf8')

count = 0

for i in pos\_str\_col:

if i < 0 or i > len(str\_file):

raise ValueError('position parameter is less than 0 or greater than the file length')

return count

except IOError:

print('{} is a bad file name'.format(file))

break

except FileNotFoundError:

print('file {}: cannot be found'.format(file))

break

#Part C

def main():

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")]

ed\_write(fn, ((2, "ABC"), (30, "DEFG")))

ed\_insert(fn, ((2, "ABC"), (10, "DEFG"))) # changed file to: 01ABC23456789DEFG0123456789

main()

I wasn’t able to finish so lots of errors on this bad boy

