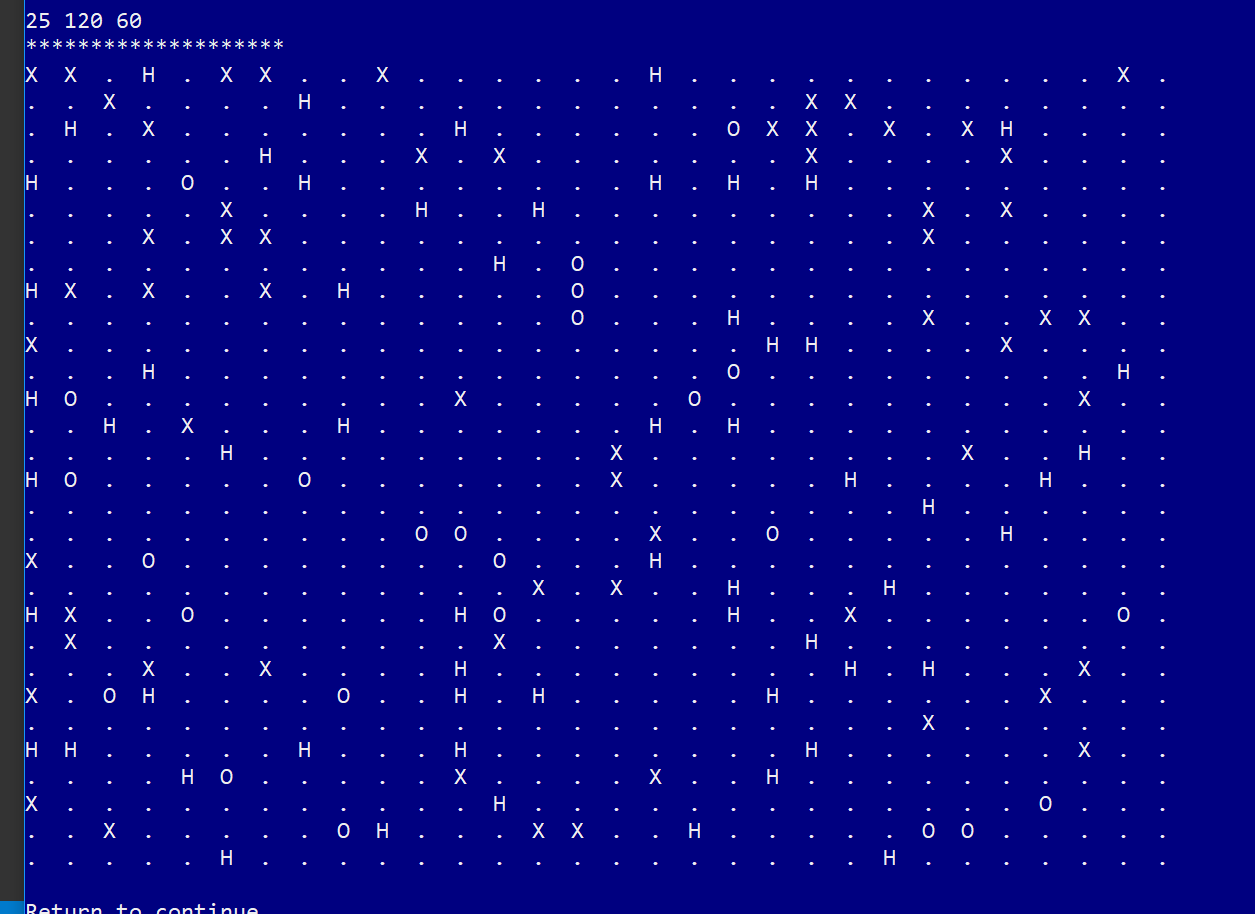
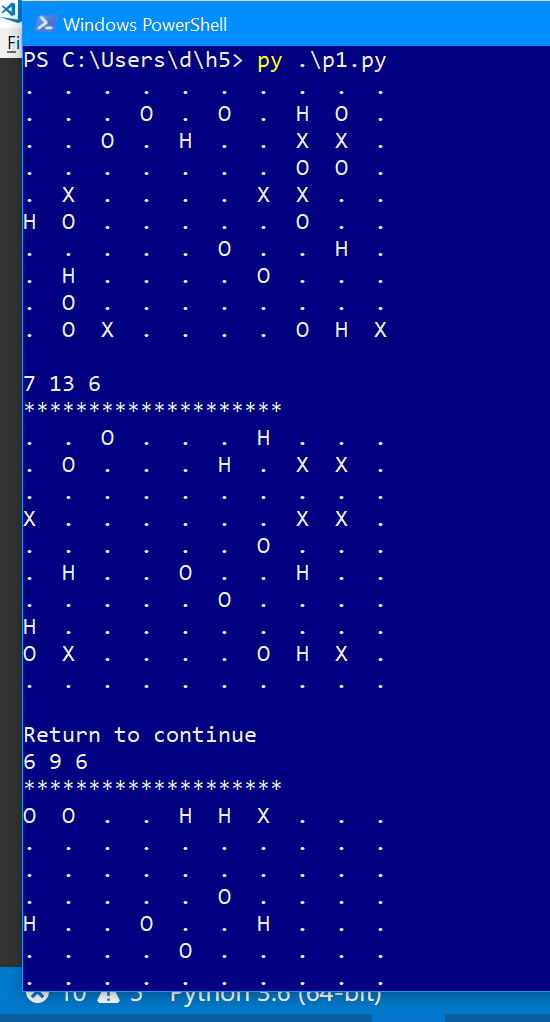
**Homework 5 Problem-1**

Pred-Prey-Humans





# Predator-Prey Simulation.

''' Adding to Island class for Human.

Tested main: Grid and worked as well at start

'''

import random

import time

import pylab

class Island (object):

"""Island n X n grid where zero value indicates not occupied.

For Human class: count\_humans(), init\_animals() adjustment

"""

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'''

# while loop continues until prey\_count unoccupied positions are found

count = 0

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)

# predator\_count

count = 0

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)

# Human\_count

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

# Added this iteration for Humans\_count

def count\_human(self):

''' count all the human 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:

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

def clock\_tick(self):

'''Prey only updates its local breed clock

'''

self.breed\_clock -= 1

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

def clock\_tick(self):

''' Predator updates both breeding and starving

'''

self.breed\_clock -= 1

self.starve\_clock -= 1

if self.starve\_clock <= 0:

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:

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):

'''Implements a hunting clock.

Inherits eat(), move(), breed(), clock\_tick()'''

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

Predator.\_\_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 hunt(self):

'''Using functions in Animal like checkgrid/remove'''

self.hunt\_clock -= 1

if self.hunt\_clock <= 0:

if not self.moved:

location = self.check\_grid(Predator)

if location:

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\_clock

self.moved = True

def main(predator\_breed\_time=6, predator\_starve\_time=3, initial\_predators=7,

prey\_breed\_time=3, initial\_prey=13, size=10, ticks=300,

human\_breedtime=9, human\_starve\_time=5, hunt\_time=5, init\_humans=6):

'''Event Loop, counting, and graphing in progress'''

# Time Quantities for ClassVariables

Predator.breed\_time = predator\_breed\_time

Predator.starve\_time = predator\_starve\_time

Prey.breed\_time = prey\_breed\_time

Human.breed\_time = human\_breedtime

Human.starve\_time = human\_starve\_time

Human.hunt\_time = hunt\_time

# Construct island

isle = Island(size, initial\_prey, initial\_predators, init\_humans)

print(isle)

# Defined outside the event loop for consistency

predator\_list = []

prey\_list = []

human\_list = []

# Event loop manages 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):

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, Human) or \

isinstance(animal, Predator):

animal.eat()

animal.move()

animal.breed()

animal.clock\_tick()

if isinstance(animal, Human):

animal.hunt()

# Display during Event-Loop

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

prey\_list.append(prey\_count)

predator\_list.append(predator\_count)

human\_list.append(human\_count)

# print out every 10th cycle

# if not i % 10:

print(prey\_count, predator\_count, human\_count)

# print the island

print('\*'\*20)

print(isle)

ans = input("Return to continue")

# Graphing after the Event-Loop

ticks = 150

pylab.plot(range(0, ticks, 2), tuple(predator\_list), label="Predators")

pylab.plot(range(0, ticks, 2), tuple(human\_list), label="Humans")

pylab.plot(range(0, ticks, 2), tuple(prey\_list), label="Prey")

pylab.legend(loc="best", shadow=True)

pylab.show()

if \_\_name\_\_ == '\_\_main\_\_':

main()