|  |  |  |
| --- | --- | --- |
| **College of Engineering**  Computer Science & Eng. Dept.  **Course:** CMP 321L Programminglanguages Lab | A picture containing logo  Description automatically generated | **Course Professor:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Summer 2022 |

**Lab 4 – Python Classes**

**Objectives:**

* Understand and implement classes in Python.
* Implement and use special methods/functions.

**Due date: End of the lab. (**Only one team member needs to submit.)

**Rules:**

(1) Usage: **You should explore and make good use of the Python features you learned in class.** (2) Scope: **You should only use those features that have been explained in detail in class.**

(3) Style: Follow standard Python programming style and conventions.

(4) Logic: Add appropriate comments to your code to explain your solution.

(Code / answers that do not follow the above specifications will be penalized.)

***Warning:* You need to come to the lab properly prepared i.e.**

(1) Make sure you have studied and understood the class material.

(2) Read the lab doc, think about the problems, and prepare questions as needed.

If you do not, completing the lab in 2.45 hours may become too much of a challenge!

**Useful resources:**

[**https://docs.python.org/3/tutorial/classes.html**](https://docs.python.org/3/tutorial/classes.html)

[**https://docs.python.org/3/reference/datamodel.html**](https://docs.python.org/3/reference/datamodel.html)

**Lab By:**

Rohan Mitra b00085023

Keshav Ramesh b00088595

Code available at: https://github.com/ro1406/CMP321Lab

**Exercise 1: Design and implement Polygon classes [8 Marks]**

Write a Polygon class that is a **sequence** of 2D points represented by **named tuples**, so that each point is given a name e.g., point ‘A’ is (4, 5). The class defines the following functions:

1. Polygon initialization, printing, length, and membership:
   1. An initializer that takes three or more points as arguments e.g., Polygon(('A',5,0), ('B',10,5), ('C',5,10), ('D',-2,8)), and defines a polygon accordingly.
   2. A function that allows using print e.g., print(poly), to output all points of a polygon in the following format: A: (5,6) -> B: (6,7) -> C: (12,15)
   3. A function that returns the number of points in the polygon, so that len(poly) works.
   4. A membership function that checks if a given point is in the polygon, or not.
2. Point insertion, retrieval, and removal, polygon comparison:
3. An insert function that inserts at the given index a new point from given name and x, y coordinates. It should throw a user defined exception ExistingPointError if the point exists. Check the name of the point as well as x and y coordinates.
4. A get function that allows retrieving a point given its name. The function should throw a user defined exception PointNotFoundError if such point does not exist.
5. A remove function that deletes a point by name. Throws PointNotFoundError.
6. A function that implements comparison, so that poly1==poly2 works. Two polygons are the same if and only if they contain the same points in the same order.
7. Polygon drawing:
   1. A function that draws a polygon on the screen, using Turtle graphics. Below is an example of code that draws a single line from points P1 to P2; adapt as necessary.

For more info about Turtle graphics, see

<https://docs.python.org/3/library/turtle.html>

import turtle

def draw\_example(p1, p2, speed=2, color='blue'):

turtle.speed(speed)

turtle.hideturtle()

turtle.penup()

turtle.goto(p1)

turtle.pendown()

turtle.color(color)

turtle.write('A')

turtle.goto(p2)

turtle.write('B')

turtle.exitonclick()

draw\_example ( (0,50), (300,150) )

PS: Just ignore any turtle.Terminator exception and run your code again.

Note: You may test the provided driver.

﻿p= Polygon(Point('A',0,50),Point('B',300,150),Point('C',400,300))

p.insert(len(p),Point('D',450,100))

p.insert(len(p),Point('E',470,200))

print("Testing getPoint: ", p.getPoint('B'))

print("Testing len: ", len(p))

print(p)

p.remove(Point('B',300,150))

print(p)

print( "Testing in ", Point('A',0,50) in p )

p1= Polygon(Point('A',0,50),Point('B',300,150),Point('C',400,300))

print(p1)

print(p1==p)

p.draw()

**Code:**

from collections import namedtuple

import turtle

Point = namedtuple('Point', 'name x y')

class ExistingPointError(Exception):

def \_\_init\_\_(self, message="Point already exists in Polygon!"):

self.message = message

super().\_\_init\_\_(self.message)

class PointNotFoundError(Exception):

def \_\_init\_\_(self, message="Point not found in Polygon!"):

self.message = message

super().\_\_init\_\_(self.message)

class LackOfArgsError(Exception):

def \_\_init\_\_(self, message="Too few arguments to define a Polygon!"):

self.message = message

super().\_\_init\_\_(self.message)

class Polygon:

def \_\_init\_\_(self,\*args):

if len(args)<3:

raise LackOfArgsError # Could also just print error message

return

self.points=list(args)

def \_\_str\_\_(self):

res=''

for pt in self.points:

if pt == self.points[-1]:

res+=f'{pt.name}: ({pt.x},{pt.y}) '

else:

res+=f'{pt.name}: ({pt.x},{pt.y}) -> '

return res

def \_\_len\_\_(self):

return len(self.points)

def \_\_contains\_\_(self,pt):

return pt in self.points

############################# part b #############################

def insert(self,index,p):

try:

if self.\_\_contains\_\_(p):

raise ExistingPointError

self.points.insert(index,p)

except ExistingPointError:

pass

def getPoint(self,name):

for pt in self.points:

if pt.name==name:

return pt

raise PointNotFoundError #Only executed if point not found

def remove(self,p):

for pt in self.points:

if pt.name==p.name:

self.points.remove(pt)

return

raise PointNotFoundError #Only executed if point not found

def \_\_cmp\_\_(self,rhs):

if len(self.points)!=len(rhs.points): return False

for pt1,pt2 in zip(self.points,rhs.points):

if not(pt1.x==pt2.x and pt1.y==pt2.y):

return False

return True

############################# part c #############################

def draw(self):

turtle.speed(50)

turtle.hideturtle()

turtle.penup()

turtle.goto((self.points[0].x,self.points[0].y))

turtle.pendown()

turtle.color('red')

turtle.write(self.points[0].name)

for pt in self.points[1:]:

turtle.goto((pt.x,pt.y))

turtle.write(pt.name)

turtle.exitonclick()

#Driver provided:

p= Polygon(Point('A',0,50),Point('B',300,150),Point('C',400,300))

p.insert(len(p),Point('D',450,100))

p.insert(len(p),Point('E',470,200))

# Test LackOfArgsError: pol=Polygon(Point('R',14,14))

# Test Existing Point Exception: p.insert(len(p),Point('A',0,50))

# Test Not Found Point Exception: print(p.getPoint('R'))

print("Testing getPoint: ", p.getPoint('B'))

print("Testing len: ", len(p))

print(p)

p.remove(Point('B',300,150))

print(p)

print( "Testing in: ", Point('A',0,50) in p )

p1= Polygon(Point('A',0,50),Point('B',300,150),Point('C',400,300))

print(p1)

print('p1==p:',p1==p)

p.draw()

**Screenshots:**

**No Errors:**

**Text

Description automatically generated**

**Turtle ouput:**

Line chart

Description automatically generated with medium confidence

**Exercise 2: Lambda with class [2 Marks]**

You are provided with an Employee class that has four attributes: ID, name, surname, rank (rank is a number between 1 to 5).

\_\_init\_\_ and \_\_str\_\_ , as well as all comparison operators <, > i.e., \_\_lt\_ and \_\_gt\_\_ (that should compare IDs) are provided.

Given Sort function is from Lab1.

Do the following,

Print all employees,

* Sorted by ID
* Sorted by name
* Sorted by rank then by ID when the rank is the same.

Note that for ID the default > operator used by sort should suffice. No lambda is required. However, for others pass a lambda that compares the relevant fields is required.

class Employee:

def \_\_init\_\_(self,ID,name,surname,rank):

self.id=ID

self.name=name

self.surname=surname

self.rank=rank

def \_\_str\_\_(self):

s="\nID = " +repr(self.id)

s += " name = "+self.name

s +=" Surname = "+self.surname

s += " Rank = " +repr(self.rank)

return s;

def \_\_lt\_\_(self,other):

return (self.id < other.id)

def \_\_gt\_\_(self,other):

return (self.id > other.id)

def sort(values, comp):

for i in range(1, len(values)):

for j in range(0, len(values)-1):

if(values[j] > values[j+1]):

values[j], values[j+1] = values[j+1], values[j];

return values

e1=Employee(10456,"AlKhan", "Ahmed", 3)

e2=Employee(10456,"AlKhan", "Ahmed", 3)

e3=Employee(23670, "Mariam", "AlSaleh", 1)

e4=Employee(11673, "Adbulla", "Malek", 2)

e5=Employee(40074, "Nour", "AlAli", 5)

lst=[e1,e2,e3,e4,e5]

**Code:**

print("By ID:")

res=sort(lst)

for x in res:print(x,end=" ")

print('-'\*90)

print("By Name:")

res=sort(lst,lambda x,y: x.name>y.name)

for x in res:print(x,end=" ")

print('-'\*90)

print("By Rank then by ID:")

res=sort(lst,lambda x,y: x.id>y.id if x.rank==y.rank else x.rank>y.rank)

for x in res:print(x,end=" ")

**Screenshot:**

A screenshot of a computer

Description automatically generated with medium confidence