Python实验报告

实验五：面向对象程序设计

姓名：余泓锷 学号：2016326603046

实验目的

了解面向对象程序设计相对与面向过程程序设计的异同点
了解类中构造、析构方法的用法
了解python对自带的方法和参数，掌握其简单的使用
了解类继承、多态和动态绑定
掌握常用类特殊方法的使用

实验内容

实验题1：

编写一个平面二维点集类，要求这个的实例对象（也就是一个点）能够计算到另一个点的距离；再编写一个函数，能够计算覆盖一系列点的最小矩形

解答：

In [1]:

```
import random
import math
class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    #计算到另一个点p的距离
    def distance(self,p):
        r1 = self.x - p.x
        r2 = self.y - p.y
        return math.sqrt(r1*r1+r2*r2)
    #重载点的小于比较运算符
    def __lt__(self,other):
        if self.x==other.x:
            return self.y<other.y
        return self.x<other.x
    #返回叉积
    def cross(self,p1,p2):
        return (p1.x-self.x)*(p2.y-self.y)-(p2.x-self.x)*(p1.y-self.y)
    #返回点积
    def dot(self,p1,p2):
        return (p1.x-self.x)*(p2.x-self.x)+(p1.y-self.y)*(p2.y-self.y)
    #返回与另一个点构成的直线参数
    def get_line(self,p):
        if self.x==p.x:
            return (1,0,-self.x)
        if self.y==p.y:
            return (0,1,-self.y)
        return (1.0/(p.x-self.x),1.0/(self.y-p.y),1.0*(self.y*p.x-self.x*p.y)/((a.x-self.x)*(a.y-se
    #返回p1到直线的距离
    def get_shadowD(self,p1,p2):
        if self==p1 or p1==p2:
            co = 1
        else:
            co = self.dot(p1,p2)/(self.distance(p1)*self.distance(p2))
        shadowD = 1.0*self.distance(p1)*math.sqrt(1-co*co)
        return shadowD
    #返回当前点与p1构成的线段在直线上的投影
    def get_levelD(self,p1,p2):
        if self==p1 or p1==p2:
            co = 0
        else:
            co = self.dot(p1,p2)/(self.distance(p1)*self.distance(p2))
        levelD = 1.0*self.distance(p1)*co
        return levelD
```

In [2]:

```
class Points:
    def __init__(self):
        self.ps = []
        self.tb = []
    def distance(self,id1=0,id2=0):
        if id1<len(self.ps) and id2<len(self.ps):
            r1 = self.ps[id1].x - self.ps[id2].x
            r2 = self.ps[id1].y - self.ps[id2].y
            return math.sqrt(r1*r1+r2*r2)
        else:
            return False
    def sort_ps(self):
        self.ps.sort()
    def get_point(self,x,y):
        p = Point(x,y)
        self.ps.append(p)
    def get_recarea(self,id0):
        le = len(self.tb)
        id1 = (id0+1)%le
        maxh = 0
        a = id1
        for i in range(0,le):
            h = self.tb[id0].get_shadowD(self.tb[i],self.tb[id1])
            if h>maxh:
                maxh = h
                a = i
        maxlk = 0
        b = id0
        for i in range(0,le):
            lk = self.tb[id0].get_levelD(self.tb[i],self.tb[id1])
            if lk>0:
                lk = 0.0
            elif lk<0:
                lk = -lk
            if lk>maxlk:
                maxlk=lk
                b = i
        maxrk = 0
        b = id1
        for i in range(0,le):
            rk = self.tb[id1].get_levelD(self.tb[i],self.tb[id0])
            if rk>0:
                rk = 0.0
            elif rk<0:
                rk = -rk
            if rk>maxrk:
                maxrk=rk
                b = i
        return 1.0*maxh*(maxlk+self.tb[id0].distance(self.tb[id1])+maxrk)
    def get_minrec(self):
        self.get_tubao()
        minarea = float("inf")
        for i in range(0,len(self.tb)):
            area = self.get_recarea(i)
            if area<minarea:
                minarea = area
        return minarea
    def get_tubao(self):
        if len(self.ps):
```

```
            self.sort_ps()
            self.tb.append(self.ps[0])
            for i in range(1,len(self.ps)):
                while len(self.tb)>1 and self.tb[-2].cross(self.tb[-1],self.ps[i])>0:
                    self.tb.pop()
                self.tb.append(self.ps[i])
            temp = len(self.tb)
            for i in range(len(self.ps)-2,0,-1):
                while len(self.tb)>temp and self.tb[-2].cross(self.tb[-1],self.ps[i])>0:
                    self.tb.pop()
                self.tb.append(self.ps[i])
            return True
        else:
            return False
```

In [3]:

```
a = Points()
a.get_point(2,2)
a.get_point(1,1)
a.get_point(2,1)
a.get_point(2,3)
a.get_point(3,2)
print('The points:',end=' ')
for b in a.ps:
    print(b.x,b.y,end=',')
print()
a.sort_ps()
print('The end of sort:',end=' ')
for b in a.ps:
    print(b.x,b.y,end=',')
print()
a.get_tubao()
print('The tubao is:',end=' ')
for b in a.tb:
    print(b.x,b.y,end=',')
print()
#for i in range(10):
#    x=random.randint(0,100)
#    y=random.randint(0,100)
#    a.get_point(x,y)
print('The min area:',a.get_minrec())
```

```
The points: 2 2,1 1,2 1,2 3,3 2,
The end of sort: 1 1,2 1,2 2,2 3,3 2,
The tubao is: 1 1,2 3,3 2,2 1,
The min area: 3.0
```

实验题2：

编写一个类Temperature，这个类只有2个类方法，进行温度转换，to_fahrenheit()用于将温度转换为华氏温度，to_celsius()将温度转换为摄氏温度

解答:

In [4]:

```
class Temperature:
    T = 25
    ch = '° C'
    def __init__(self, temperature=25):
        Temperature.T = temperature
        Temperature.ch = '° C'
    @classmethod
    def to_fahrenheit(cls):
        if cls.ch=='° C':
            cls.ch = '° F'
            cls.T = 1.0*cls.T*9/5+32
        print('当前温度为：',cls.T,cls.ch)
    @classmethod
    def to_celsius(cls):
        if cls.ch=='° F':
            cls.ch = '° C'
            cls.T = 1.0*5/9*(cls.T-32)
        print('当前温度为：',cls.T,cls.ch)
t = Temperature(30)
t.to_fahrenheit()
t.to_celsius()
t.to_celsius()
t.to_fahrenheit()
```

```
当前温度为： 86.0 ° F
当前温度为： 30.0 ° C
当前温度为： 30.0 ° C
当前温度为： 86.0 ° F
```

实验题3：

编写一个电梯类，能够模拟日常电梯的各种操作。

解答：

In [9]:

```
import time
class Elevator:
    def __init__(self,layer,first_layer=1):
        self.max_layer = layer;                 #电梯最高楼层
        self.option = 'up'                      #电梯当前操作
        self.door = 'close'                     #电梯门状态
        self.cur_layer = first_layer            #当前电梯所在楼层
        self.up_queue = []                      #电梯上行楼层队列
        self.down_queue = []                    #电梯下行楼层队列
        self.switch = 'run'                     #电梯状态
    #电梯信息显示
    def info(self,flag=False):
        print('当前电梯在{0}层，电梯正在{1}!'.format(self.cur_layer,self.option))
    #电梯停止运作
    def stop(self):
        self.switch = 'stop'
        print('电梯停止运行')
    #电梯门打开
    def open_door(self):
        self.door = 'open'
        print('电梯门打开')
    #电梯门关闭
    def close_door(self):
        self.door = 'close'
        print('电梯门关闭')
    #电梯等待
    def wait(self):
        self.open_door()
        print('当前电梯在{0}层，电梯正在等待!'.format(self.cur_layer))
        time.sleep(5)
        self.close_door()
    #楼层加入队列
    def push_queue(self,layer,option=None):
        if layer<self.cur_layer:
            if not layer in self.down_queue:
                self.down_queue.append(layer)
                self.down_queue.sort(reverse=True)
        elif layer>self.cur_layer:
            if not layer in self.up_queue:
                self.up_queue.append(layer)
                self.up_queue.sort()
        else:
            if option==self.option:
                self.wait()
            else:
                if option=='up':
                    if not layer in self.up_queue:
                        self.up_queue.append(layer)
                        self.up_queue.sort()
                else:
                    if not layer in self.down_queue:
                        self.down_queue.append(layer)
                        self.down_queue.sort(reverse=True)
    #电梯运行
    def run(self):
        while self.switch=='run':
            que = getattr(self,self.option+'_queue')
            if not len(self.up_queue) and not len(self.down_queue):
                if self.cur_layer==1:
```

```
                pass
            else:
                self.push_queue(1)
        elif not len(self.up_queue) and len(self.down_queue):
            self.option = 'down'
        elif len(self.up_queue) and not len(self.down_queue):
            self.option = 'up'
        if self.option=='up':
            inc = 1
        else:
            inc = -1
        while len(que):
            flag = (que[0]==self.cur_layer)
            if flag:
                que.pop(0)
                self.wait()
            else:
                self.info()
                self.cur_layer += inc
            time.sleep(1)
```

In [10]:

```
import threading
def init_elevator(building_layers):
    e = Elevator(building_layers)
    t = threading.Thread(target = e.run)
    t.setDaemon(True)
    t.start()
    return (e,t)
```

In [13]:

```
min_layer = 1       #电梯最低楼层
max_layer = 18      #电梯最高楼层
myelevator,ctl_thread = init_elevator(max_layer)
while True:
    str1 = input('Input the layer: ')
    try:
        layer = int(str1)
    except Exception:
        if str1=='quit':
            myelevator.stop()
            ctl_thread.join()
            break
        else:
            print('invalid input',str1)
            continue
    if layer not in range(min_layer,max_layer+1):
        continue
    myelevator.push_queue(layer)
```

```
Input the layer: 8
当前电梯在1层，电梯正在up!
当前电梯在2层，电梯正在up!
当前电梯在3层，电梯正在up!
当前电梯在4层，电梯正在up!
Input the layer: 5
当前电梯在5层，电梯正在up!
当前电梯在6层，电梯正在up!
当前电梯在7层，电梯正在up!
电梯门打开
当前电梯在8层，电梯正在等待!
Input the layer: 2
电梯门关闭
当前电梯在8层，电梯正在down!
当前电梯在7层，电梯正在down!
当前电梯在6层，电梯正在down!
电梯门打开
当前电梯在5层，电梯正在等待!
电梯门关闭
当前电梯在5层，电梯正在down!
当前电梯在4层，电梯正在down!
当前电梯在3层，电梯正在down!
电梯门打开
当前电梯在2层，电梯正在等待!
Input the layer: 4
电梯门关闭
当前电梯在2层，电梯正在up!
当前电梯在3层，电梯正在up!
电梯门打开
当前电梯在4层，电梯正在等待!
电梯门关闭
当前电梯在4层，电梯正在down!
当前电梯在3层，电梯正在down!
当前电梯在2层，电梯正在down!
电梯门打开
当前电梯在1层，电梯正在等待!
电梯门关闭
Input the layer: quit
电梯停止运行
```