Vidzeme University of Applied Sciences

**Faculty of Engineering**

Python OOP and Modelling

group 01

practical work #6

Valmiera, 2022

**Table of contents**

[1 Activity Diagram 4](#_Toc94044448)

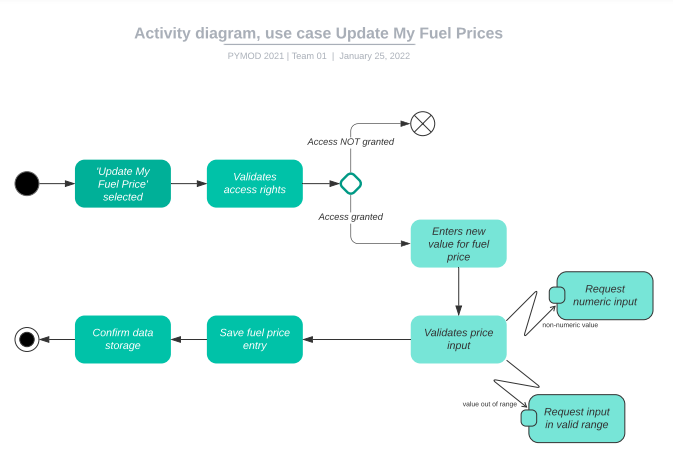
[2 Drawing shapes world 5](#_Toc94044449)

|  |  |  |  |
| --- | --- | --- | --- |
| **Document versions** | | | |
| Version | Status / Changes | Date | Author |
| 1.0 | Initial document release | 25.01.2022 | S.Grosberga |
| 1.1 | Python task documentation added | 25.01.2022 | A.Jenerts |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Contacts and responsible (-s)** | | | |
| Name Surname | Department | Position | Contact information (e-mail) |
| Signe Grosberga | Group 01 | Member | signe.grosberga@va.lv |
| Edmunds Sprūdžs | Group 01 | Member | edmunds.sprudzs@va.lv |
| Andris Jenerts | Group 01 | Member | andris.jenerts@va.lv |

# Activity Diagram

Activity Diagram was developed for use case “Update my Fuel Prices”. Activity diagram contains two exception cases for non-numeric data entry and price out of 'reasonable' value range. Also, the activity flow is stopped for user not authorized to edit or update fuel price register.



1.image *Activity Diagram, use case Update My Fuel Prices*

# Drawing shapes world

Storing Point coordinate pair as tuple can be implemented modifying only Point class. Old variables for x and y coordinate is replaces with single self.\_\_pos(x,y). This is tuple with two elements. Point class still expects two parameters: one for x and second for y coordinate.

* First element (index 0) is x coordinate
* Second element (index 1) is y coordinate

Accordingly to made modification Point class methods that use coordinates are modified using self.\_\_pos[0] to reference x coordinate and self.\_\_[1] to reference y coordinate.

class Point(GeometricObject):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.\_\_pos = (x,y)

    def getCoord(self):

        return (self.\_\_pos[0], self.\_\_pos[1])

    def setPos(self,x,y):

        self.\_\_pos = (x, y)

    def getX(self):

        return self.\_\_pos[0]

    def getY(self):

        return self.\_\_pos[1]

    def \_draw(self, turtle):

        turtle.goto(self.\_\_pos[0], self.\_\_pos[1])

        turtle.dot(self.getWidth(), self.getColor())

New class Polygon is created to ease drawing polygons. This is class is not abstract class and can be used to draw shapes directly. Polygon class expects single parameter - list of Point objects.

class Polygon(GeometricObject):

    """

    Polygon class that conects all points passed as list

    """

    def \_\_init\_\_(self, points:List[Point]):

        super().\_\_init\_\_()

        self.\_\_points = points

    def \_draw(self, turtle):

        turtle.color(self.getColor())

        turtle.width(self.getWidth())

        turtle.up()

        turtle.goto(self.\_\_points[0].getCoord())

        turtle.down()

        for point in self.\_\_points:

            turtle.goto(point.getCoord())

        turtle.goto(self.\_\_points[0].getCoord())

        turtle.up()

Polygon class \_draw methods main goal is to connect all point in \_\_points list. It is done executing such steps:

* lifting pen (for moving to star position (first point))
* moving to first point
* putting pen down
* iterating trough all the points and moving to their location
* going to the first point to close the polygon
* lifting pen up

class CenterPolygon(Polygon):

    """

    Specify center coordinate, number of points and radius

    """

    def \_\_init\_\_(self, center:Point, radius, n:int ):

        self.\_\_r = radius

        self.\_\_n = n

        self.\_\_center = center

        self.\_\_centerAngle = 360/n

        self.\_\_points = []

        for n in range(self.\_\_n):           # iterate trough all vertices

            self.\_\_points.append(Point(     # add new point to the list

                # x coordinate of the point

                (math.sin(math.radians(self.\_\_centerAngle\*n))\*self.\_\_r + self.\_\_center.getX()),

                # y coordinate of the point

                (math.cos(math.radians(self.\_\_centerAngle\*n))\*self.\_\_r + self.\_\_center.getY())

                ))

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

Center polygon class is most complicated of Polygon convenience classes. This class expects two arguments in constructor.

* center - Point object that specifies center of the polygon to be drawn
* radius - number of vertices for the polygon

Some basic trigonometry is utilized to calculate coordinates of the vertices for the polygon. To perform mathematical functions python Math module is used.

After coordinates for all vertices are calculated and saved in single list, this list is passed to Polygon parent class init function.

Triangle class is similar to Polygon class. Only difference is that it expects three points fot construction. These points are passed to Polygon constructor as list to draw triangle.

class Triangle(Polygon):

    """

    Triangle is same as polygon

    specify 3 points

    """

    def \_\_init\_\_(self, p1:Point, p2:Point, p3:Point):

        super().\_\_init\_\_([p1, p2, p3])

Rectangle class expects two points needed to define rectangle. First one for the bottom left point and second for top right.Using these coordinates remaining two point coordinates can be calculated and rectangle drawn.

class Rectangle(Polygon):

    """

    Specify bottom left and top right points

    to draw a rectangle

    """

    def \_\_init\_\_(self, x0y0:Point, x1y1:Point):

        super().\_\_init\_\_(

            [

                x0y0,                               # bottom left

                Point(x0y0.getX(), x1y1.getY()),    # top left

                x1y1,                               # top right

                Point(x1y1.getX(), x0y0.getY())     # bottom right

            ]

        )

Octagon class ir child of centerPolygon class. Octagon expects center point (Point object) and radius.

class Octagon(CenterPolygon):

    """

    Specify radius and center location

    it is the same center polygon only with specified

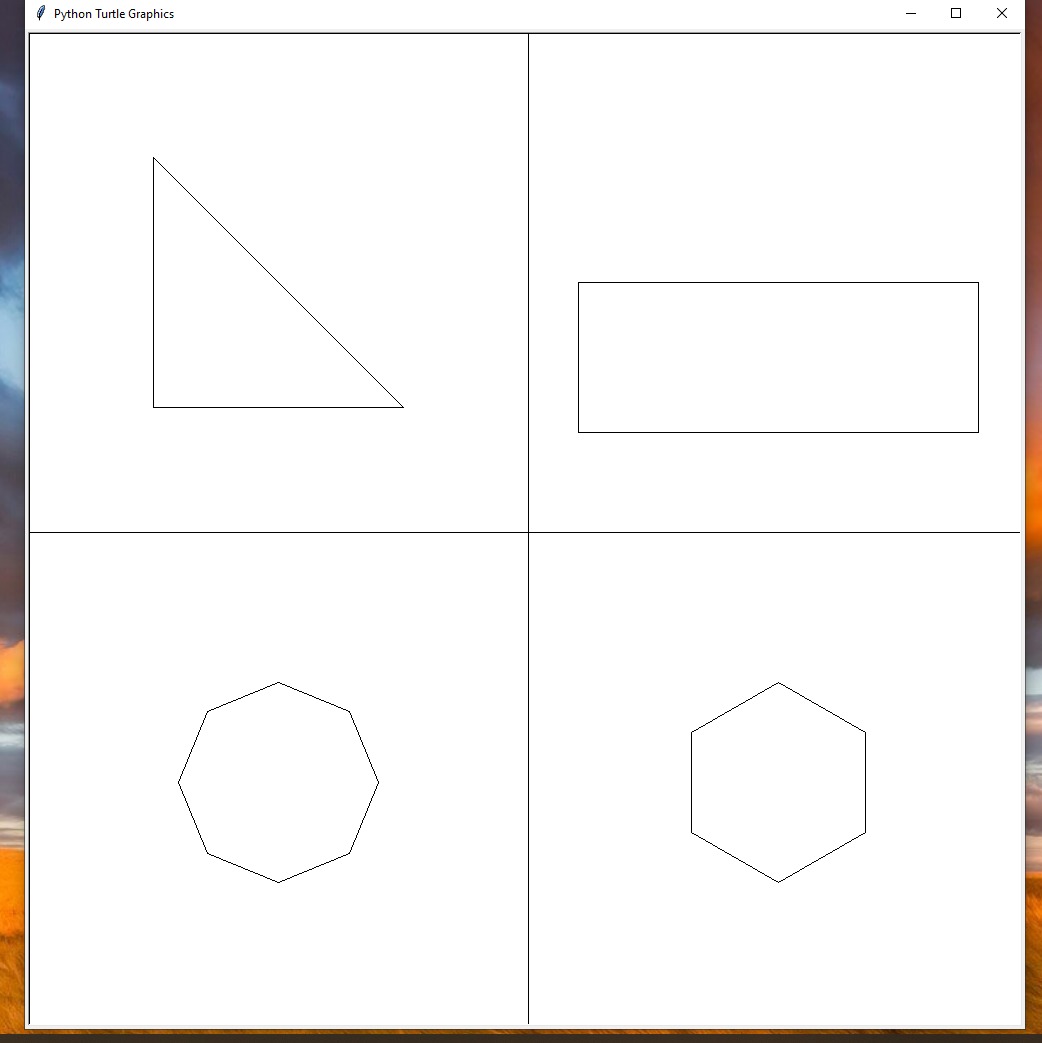
    number of vertices.

    """

    def \_\_init\_\_(self, center: Point, radius):

        super().\_\_init\_\_(center, radius, 8)

Test is written so that Line objects are used to create + shape. In I quadrant rectangle is drawn, in II triangle, in III octagon, IV quadrant is used for centerPolygon that can have random number of vertices (between 3 and 7)



2.image *The shapes world simulation with written Polygon convenience classes*