VIDZEME UNIVERSITY OF APPLIED SCIENCES

VIDZEME UNIVERSITY OF APPLIED SCIENCES

FACULTY OF ENGINEERING

PYTHON OOP AND MODELLING

GROUP 01

PRACTICAL WORK #2

VALMIERA, 2021

Table of contents

Document versions			
Version	Status / Changes	Date	Author
1.0	2.1. task documentation added	05.12.2021	A.Jenerts
1.1	2.2. task documentation added	06.12.2021	S.Grosberga
1.2	UML diagram and its description added	06.12.2021	E.Sprūdžs

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

1 Use case diagram

Several UML and Use case online resources were studied to learn more about the subject and complete the assignment. [6,7]

Use case diagram for a system to assist drivers with selecting the best fuel price was discussed, agreed upon and designed. Three potential human actors where identified - Driver, Fuel Station/Network operator and System Administrator. Fourth actor(-s) in the form or remote systems that can supply the necessary fuel price data was also identified. The relevant actions were assigned and relations between Update actions where documented.

An online tool Lucid containing specialized toolset for Use case diagrams was used.[8]

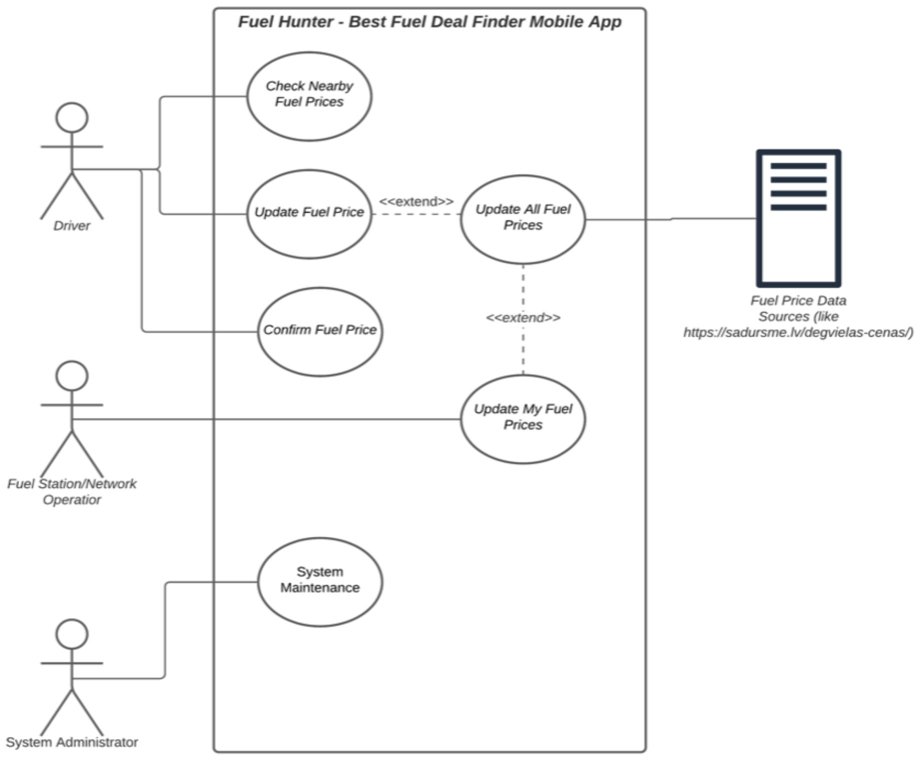

Image 1 UML Use case diagram

2 Drawing n-pointed star with turtle graphics

Task is to make function that draws n-pointed star using Python turtle module. It is stated that n can be any number that is odd. It should be clear that these number can be only integers and no less than 5 because it is impossible to draw star with less than 5 vertices, 3 vertices form triangle that is polygon.

2.1 Geometry overview

To better understand process of drawing stars we tried to draw different variants of them on the paper. Drawing 5-point stars are straight forward but while drawing 7-point star it becomes clear that is it possible to draw such star in multiple ways. Both ways are shown in 1. image below.

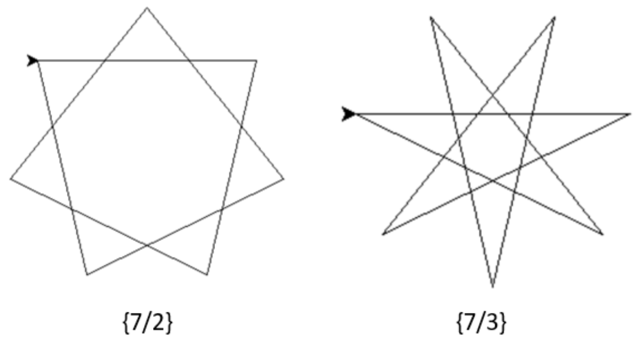

Image 2 Two variants of 7-pointed star

Doing a bit further research team found out that there is special notation for such phenomenon called Schläfli symbols.[1] Not going into deep details this notation consists of two parts:

$$p - number\ of\ vertices$$

$$q - \ turning\ number$$

In general form Schläfli symbol for star looks like shown below. For this work p will be replaced with n because of name of the task “n-pointed star”

$$\{p/q\}$$

To better understand q term one should loop at image below, where turning number is visualized

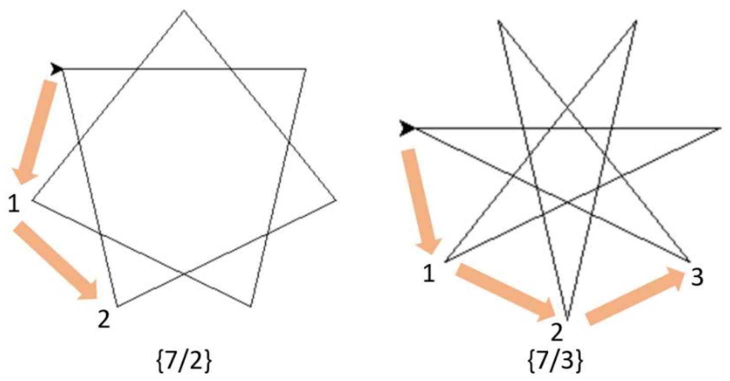

Image 3 Turning number illustration

It is important to look at all possible variants of 7-pointed star to derive general formula for finding how many possibilities there are to draw a star in different ways. In the table below you can see all possible variant noted in Schläfli symbols. To each symbol description is provided that explain is such combination is possible or not.

No.	Schläfli Symbol	Notes
1	$\{7/1\}$	This would result in polygon and polygons are not considered stars
2	$\{7/2\}$	First variant of the star
3	$\{7/3\}$	Second variant of the star
4	$\{7/4\}$	Result in the same variant as $\{7/3\}$
5	$\{7/5\}$	Results in the same variant as $\{7/2\}$
6	$\{7/6\}$	Same as $\{7/1\}$ – not a star
7	$\{7/7\}$	This is a point

It becomes clear that there are n variants for n-pointed stars, but only some of them results in stars, always 2 of them results in polygons and one is a point. Rest of the combinations are valid but are paired meaning that two combinations produce the same variant of the star e.g., $\{7/2\}$ and $\{7/5\}$. From these observations it is possible to create general formula for calculating how many stars are possible to draw:

$$v = \frac{n-3}{2}$$

$$where: n - points\ of\ the\ star$$

NOTE: this formula is intended for odd number n as intended for this task.

There is formula for calculating star angle sum that will be used in our Python function to calculate turning angle. [2]

$$\sum \alpha_0 = (n-2q)180°, \qquad where$$

$$\alpha_0 - star\ angle$$
$$n - point\ of\ the\ star$$
$$q - turning\ number$$

2.2 Turning angle calculation

To implement angle sum formula, we must look at the way how turtle is moving in our program. Is starts by moving straight and then comes to the vertex of the star. This is the point where it needs to do the turning. This is how turning angle is calculated:

1) Calculate angle of the vertex using star angle sum formula and number of the vertices.

$$\alpha_0 = \frac{(n-2q)180}{n}$$

2) Turning angle β van be calculated using basic geometry shown in image 3.

$$\beta = 180 - \alpha_0$$

$$\beta = 180 - \frac{(n-2q)180}{n}$$

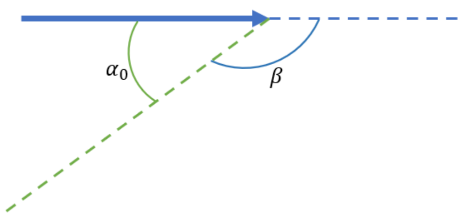

Image 4 Turning angle geometry

Simplifying this equation as shown below it is possible to get simple formula for turning angle β

$$\beta = 180 - \frac{(n-2q)180}{n} \rightarrow \beta n = \frac{180n}{n} - \frac{(n-2q)180}{n}$$

$$\beta n = 180n - 180(n-2q) \rightarrow \beta n = 180(n-n+2q)$$

$$\beta n = 180 * 2q \rightarrow \boldsymbol{\beta = \frac{360q}{n}}$$

2.3 Repeating patterns

There are two possible types of stars. Ones that can be drawn with out lifting pen from the paper and ones that need lifting of the pen. First odd number point star that need lifting pen is $\{9/3\}$ star. This star consists of tringles rotated at different angles and stacked on top of each other.

Authors made wrong assumption that stars it is possible to identify such stars when n divides with q without reminder. But this is not true as we fond out later. Starts that need pen lifting action can be identified if both n and q has common divider. For example, before mentioned $\{9/3\}$ stars both elements divide by 3, such star consists of 3 triangles (because common divider is 3 $(defines\ shape)$ and $nn/q \rightarrow 9/3 = 3\ (defines\ count)$. As this was identified only later into development, due to exponentially rising code size and difficulty it was decided to develop algorithm only for firstly made assumption.

In the image below you can see previously mentioned $\{9/3\}$ star that will be used to explain geometry behind drawing such star.

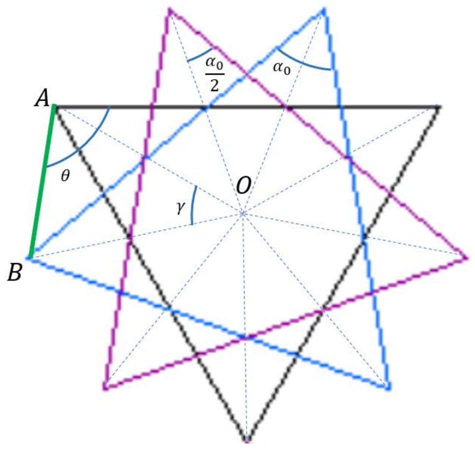

Image 5 {9/3} star

Steps for drawing such star:

a) First triangle is drawn as usually, using same turning angle calculation as previously described
b) Once first triangle (the black one) is drawn, pen is lifted, turtle should rotate at point A so that it faces B, in other words rotate right by angle θ_f

   Angle theta is calculated based on two angles:
   - Vertex angle α_0 – calculation explained previously
   - angle γ that can be calculated with formula:

$$\gamma = \frac{360}{n}$$

Knowing these two angles is tis possible to calculate both angle θ_f and distance AB. Firstly, small part of the angle θ_f is calculated by knowing that AO is bisectriz of vertex angle:

$$\angle OAM = \frac{\alpha_0}{2}$$

Secondly $\angle BAO$ can be calculated in such way:

$$\angle BAO = \frac{180 - \gamma}{2}$$

Adding these angles together we get angle that turtle needs to turn to face B point:

$$\theta_f = \frac{\alpha_0}{2} + \frac{180 - \gamma}{2}$$

c) distance AB is chord and can be calculated using general chord length formula using radius (fixed value in our case) and center angle γ:

$$AB = r * sin\left(\frac{\gamma}{2}\right)$$

d) At the point B turtle needs to point to the next vertex at point N, meaning it needs to turn by angle θ_b to the left, this angle can be calculated in similar fashion as θ_f:

$$\theta_b = 180 - \angle ABO + \frac{\alpha_0}{2}$$

$$\theta_b = 180 - \frac{180 - \gamma}{2} + \frac{\alpha_0}{2}$$

$$\theta_b = \frac{(180 + \gamma)}{2} + \frac{\alpha_0}{2}$$

e) After this process can restart at step and repeat until all star is drawn.

It is important to note that same algorithm woks for any n-pointed polygons that will overlap and form star e.g., {15/3} – 3 overlapping pentagons. This **does not work** for stars that is drawn by overlapping other stars e.g., {15/6}

2.4 Star drawing function

Simple function that takes turtle object, number of points and turning number is written to draw a star. This function does not handle any user input errors. Error handling is done outside of the function. As side length is not specified in task it is hard coded and cannot be changed without altering the code.

As previously explained some math functions are needed to calculate different angles and lengths of the sides, for that reason python math module was imported and used.

Full program can be found in first appendix.

```python
def drawStar(thisTurtle: turtle.Turtle, n:int, q:int):
    starRadius = 100                                                  #this is fixed number
    vertexAngle = (180*(n-2*q))/n                                     #angle sum divided by no. of vertices
    centerAngle = 360/n
    turnAngle = 360*q/n                                               #calculates turning angle at the vertex
    sideLength = starRadius * math.sin(math.radians(centerAngle*q)/2) #calculates length of the edge

    liftAngle = (vertexAngle/2 + (180-centerAngle)/2)                 #once pen is lifted turn this angle
    liftDistance = starRadius * math.sin(math.radians(centerAngle/2)) #this is distance to the next point
    liftBackAngle = (180+centerAngle)/2 + vertexAngle/2         #angle to turn back and start next figure

    if(n % q != 0):
        # these stars can be drawn without "lifting pen"
        for point in range(n):
            thisTurtle.forward(sideLength)                     #draw a line
            thisTurtle.right(turnAngle)                        #turn to draw next line
    else:
        for lift in range(q):
            for point in range(int(n/q)):
                thisTurtle.forward(sideLength)                 #draw a line
                thisTurtle.right(turnAngle)                    #turn to draw next line

            thisTurtle.penup()                                 #lift up pen
            thisTurtle.right(liftAngle)                        #turn to the next point direction
            thisTurtle.forward(liftDistance)                   #go to the next point
            thisTurtle.left(liftBackAngle)                     #turn back to draw next edge of the star
            thisTurtle.pendown()
```

2.5 User inputs and error handling

Simple input validation is done to ensure that:

a) Numeric values are entered
b) Entered numeric values are integers
c) Entered n is odd number
d) Entered n corresponds to star – meaning that it is at least 5

In all these cases program prompts that user has entered something wrong and asks to enter new value. Program also doesn’t close automatically after stars are drawn, it asks user to press any key and then exits.

3 Drawing mathematical function using turtle graphics

Tasks is to create program that draws simple mathematical function $y = \frac{x}{2} + 5$ using python turtle module. It is noted that x value range should be at least 150 units.

No user input is necessary for this task. Two main actions are identified:

a) Drawing x and y axis
b) Plotting function on the axis

3.1 Drawing axis for the graph

A function that takes turtle object, distance, and mark was written to draw the axis for the graph. Function code draws single axis of the graph distance unit long in both directions (positive and negative).

While drawing the axis, turtle sets markings for the axis every unit. This slows down the function but by the teams' thoughts improves readability of the graph.

```
def drawAxis(turtle, distance, mark):
    position = turtle.position()
    turtle.pendown()

    for _ in range(0, distance, mark):
        turtle.forward(mark)
        turtle.dot() #makes runtime slower, but makes axis easier to read

    turtle.setposition(position)

    for _ in range(0, distance, mark):
        turtle.backward(mark)
        turtle.dot()
```

Function is called twice to draw both axis, between these function calls turtle is rotated 90° using turtle method *setheading().*

3.2 Plotting the function

A separate function was written to plot point of the given mathematical function $y = \frac{x}{2} + 5$ For each x in the range, turtle goes to the next point in the coordinate system within the range. Whole function is provided below. Image 5 shows plotted graph by the function. Appendix 2 contains full program

```
def drawFunction(turtle, range):
    turtle.penup()
    turtle.pencolor("red")

    for x in range:
            y = x / 2 + 5
            t.goto(x, y)
            turtle.pendown()
```

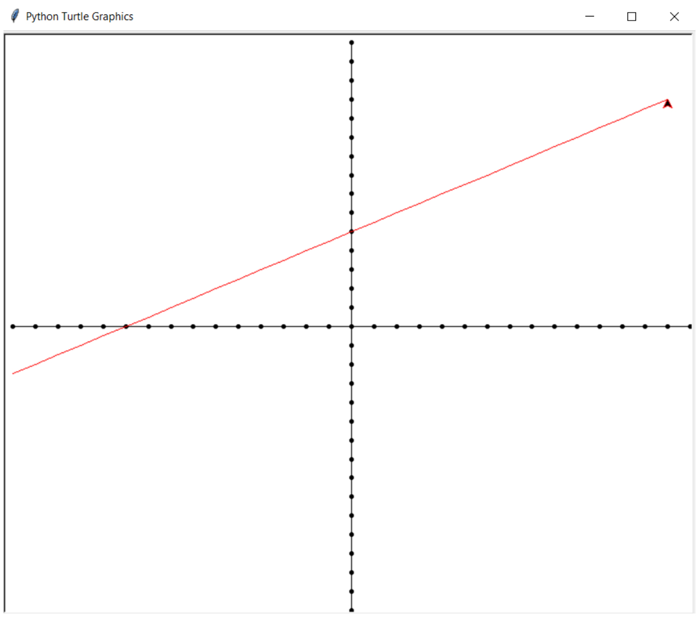

Image 6 Plotted $y = \frac{x}{2} + 5$ function

3.3 Measuring programs runtime

Timeit module was used to measure runtime of the program with different range of points provided. Code example of runtime measuring method is shown below

```
import timeit
start = timeit.default_timer()
stop = timeit.default_timer()
print('Time: ', stop - start)
```

Table below summarizes how point count and marking influence runtime of the program

Range	t, s (/w markings)	t, s (/wo markings
-150;150	66.797	16.484
-50;50	16.741	7.149
-15;15	9.094	6.220

4 Used resources

1. Tychonievich, L. (2012). Schläfli Symbols. Retrieved 5 December 2021, from https://www.cs.virginia.edu/~lat7h/blog/posts/219.html
2. Alsina, C., & Nelsen, R. (2010). *Charming proofs* (pp. 60-61). Washington (DC): M.A.A.
3. Turtle — Turtle graphics — Python 3.10.0 documentation. Retrieved 5 December 2021, from https://docs.python.org/3/library/turtle.html
4. math — Mathematical functions — Python 3.10.0 documentation. Retrieved 6 December 2021, from https://docs.python.org/3/library/math.html
5. timeit — Measure execution time of small code snippets — Python 3.10.0 documentation. Retrieved 6 December 2021, from https://docs.python.org/3/library/timeit.html
6. User, S. System use case diagrams. Retrieved 6 December 2021, from https://www.togaf-modeling.org/models/application-architecture/system-use-case-diagrams.html
7. Use Case Diagram Tutorial (Guide with Examples). Retrieved 6 December 2021, from https://creately.com/blog/diagrams/use-case-diagram-tutorial/
8. Lucid visual collaboration suite. Retrieved 6 December 2021, from https://lucid.app

5 APENDIX 1

2.1. task python program

```
# Second practical work: n-pointed star
# 2021.12.05
# Group 1


import turtle  # import turtle graphics module
import math
# turtle module documentation:
# https://docs.python.org/3/library/turtle.html


def drawStar(thisTurtle: turtle.Turtle, n:int, q:int):
    """ Draw a n-pointed star using q turing number.

     - thisTurtle - turtle module Turtle class object
     - n          - points of the star
     - q          - turning number (see Schläfli symbol for more info)
    """
    starRadius = 100                                                  #this is fixed number
    vertexAngle = (180*(n-2*q))/n                                     #angle sum divided by no. of vertices
    centerAngle = 360/n
    turnAngle = 360*q/n                                               #calculates turning angle at the vertex
     sideLength = starRadius * math.sin(math.radians(centerAngle*q)/2)      #calculates length of the edge if
fixed R

    liftAngle = (vertexAngle/2 + (180-centerAngle)/2)                      #once pen is lifted turn this angle
    liftDistance = starRadius * math.sin(math.radians(centerAngle/2))     #this is distance to the next point
    liftBackAngle = (180+centerAngle)/2 + vertexAngle/2             #angle to turn back and start next figure

    #for debugging purposes
    print("radius: %d vertex: %f turning: %f lift: %f liftb: %f side: %f liftSide: %f"%(starRadius, vertexAngle,
turnAngle, liftAngle, liftBackAngle, sideLength, liftDistance))

    if(n % q != 0):
        # these stars can be drawn without "lifting pen"
        for point in range(n):
            thisTurtle.forward(sideLength)                     #draw a line
            thisTurtle.right(turnAngle)                        #turn to draw next line
    else:
        for lift in range(q):
            for point in range(int(n/q)):
                thisTurtle.forward(sideLength)                 #draw a line
                thisTurtle.right(turnAngle)                    #turn to draw next line

            thisTurtle.penup()                                 #lift up pen
            thisTurtle.right(liftAngle)                        #turn to the next point direction
            thisTurtle.forward(liftDistance)                   #go to the next point
            thisTurtle.left(liftBackAngle)                     #turn back to draw next edge of the star
            thisTurtle.pendown()                               #prepare to draw

while True:
    try:

        vertices = int(input('Enter star vertices count: '))   #convert user input to int
        if((vertices % 2) == 0):
            print("I can draw only stars with odd number vertices")
            continue                                           #resume while loop to ask again
        if(vertices < 5):
            print("Star has at least 5 vertices")
            continue
        break                                                  #all test passed, break out
    except:
        print("Error, please enter integer value")             #most likely entered float or non-numeric
        continue
```

```python
# knowing how many pointed star user wants to draw
# we calcaulate all possible variants for this star and
# provide them so that user can choose from them

variant_count = int((vertices - 3)/2)                               # calculates and stores star variants
turningNumber = None                                                # this will be determined
drawAll = False                                                     # if user wants to see all variants
if(variant_count > 1):
    print("""It is possible to draw %d-pointed star in %d differnet ways.
    They are listed below in Schläfli notation"""%(vertices, variant_count))

    selected_variant = None                                         # user will select one of the variants
    for variant in range(variant_count):
        print("%d: "%(variant+1), end="")
        print("{%d,%d}"%(vertices,(2+variant)))
    print("%d: all variants"%(variant_count+1))

    while True:
        try:
            selected_variant = int(input("Enter number of the variant you want to draw: "))
            if(selected_variant < 1  or selected_variant > (variant_count)):
                if(selected_variant == (variant_count +1)):
                    print("drawing all variants")
                    drawAll = True
                    break

                print("Error, please enter valid value")
                continue

            turningNumber = selected_variant + 1                                        # 1st variant q = 2, 2nd q
= 3 etc
            break
        except:
            print("Error, please enter valid value")
            continue


else:
    turningNumber = 2                                                           # 5 pointed star



myTurtle = turtle.Turtle()      # create turtle object instance
myTurtle.speed(5)               # sets turtle speed
myTurtle.pencolor("#33cc33")    # color of the line
myTurtle.penup()
myTurtle.setposition(-400,0)
myTurtle.pendown()

if(drawAll):
    for variant in range(variant_count):
        turningNumber = variant+2
        drawStar(myTurtle, vertices, turningNumber)

        myTurtle.penup()
        myTurtle.setposition(myTurtle.pos()[0]+100,0)
        myTurtle.setheading(0)
        myTurtle.pendown()
else:
    drawStar(myTurtle, vertices, turningNumber)

input("Press any key to exit")
```

6 APENDIX 2

2.2 task python program

```
# Second practical work: drawing function x/2+5
# 2021.12.05
# Group 1

import turtle
import timeit

#starts timer, when program is started
start = timeit.default_timer()

wn=turtle.Screen()
#sets screen width and height; later used to draw axis
w=50
h=50
wn.setworldcoordinates(-w, -h, w, h)
t=turtle.Turtle()
t.position()
t.heading()
#function to draw axis
def drawAxis(turtle, distance, mark):
    position = turtle.position()
    turtle.pendown()
    turtle.dot()

    for i in range(0, distance, mark):
        turtle.forward(mark)
        turtle.dot() #makes rundown slower, but makes axis easier to read

    turtle.setposition(position)

    for i in range(0, distance, mark):
        turtle.backward(mark)
        turtle.dot()
#draws x-axis
t.penup()
t.home()
drawAxis(t, w, 1)
#draws y-axis
t.penup()
t.home()
t.setheading(90)
drawAxis(t, h, 1)
#function to draw y = x / 2 + 5
def drawFunction(turtle, range):
    turtle.penup()
    turtle.pencolor("red")

    for x in range:
            y = x / 2 + 5
            t.goto(x, y)
            turtle.pendown()
#draws mathematical function in given range
drawFunction(t, range(-50, 50))
#stops the timer and prints out runtime
stop = timeit.default_timer()
print('Time: ', stop - start)

input("Press any key to exit")
```