# Exercise 1

'''This program draws a circle'''

import turtle

import math

'''Draws circle'''

def drawCircle(t, radius, xPosition, yPosition):

t.speed(10)

t.up()

t.goto(xPosition, yPosition)

t.right(180)

t.forward(radius)

t.right(90)

t.down()

distance = 2 \* math.pi \* radius / 120

for count in range(120):

t.forward(distance)

t.right(3)

return distance

'''Main function of the program'''

def main():

t = turtle.Turtle()

t.home

xPosition = int(input("Enter X position: "))

yPosition = int(input("Enter Y position: "))

radius = int(input("Enter radius: "))

distance = drawCircle(t, radius, xPosition, yPosition)

distanceTravelled = (distance \* 120) + radius

print("The turtle travelled " + str((distanceTravelled)) + " pixels.")

turtle.done()

if \_\_name\_\_ == "\_\_main\_\_":

main()

This is the code we came up with for this exercise. Below are the outputs:

1.

Enter X position: -100

Enter Y position: -100

Enter radius: 300

Circle

Description automatically generated with medium confidence

The turtle travelled 2184.955592153876 pixels.

2.

Enter X position: 50

Enter Y position: 50

Diagram, venn diagram

Description automatically generatedEnter radius: 200

The turtle travelled 1456.6370614359173 pixels.

3.

Enter X position: 23

Enter Y position: 45

A picture containing shape

Description automatically generatedEnter radius: 75

The turtle travelled 546.238898038469 pixels.

4.

Enter X position: 67

Enter Y position: 234

A picture containing circle

Description automatically generatedEnter radius: 321

The turtle travelled 2337.9024836046474 pixels.

Below is the code for the Turtle.circle() comparison. The first code and outputs will be for using Turtle.circle(), then it will be our code replicating it along with outputs matching the same radius parameter as Turtle.circle().

'''This program uses turtle's .circle method to draw a circle'''

import turtle

t = turtle.Turtle()

def main():

t.home

radius = int(input("Enter radius: "))

t.speed(1)

t.circle(radius)

turtle.done()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Outputs

1.

Shape, circle

Description automatically generatedEnter radius: 500

2.

Enter radius: 20Shape, circle

Description automatically generated0

3.

Enter radius: 50

A picture containing scatter chart

Description automatically generated

4.

Enter radius: 89

A picture containing text, screenshot

Description automatically generated

This was our code:

'''This program replicates turtle's .cirlce method to draw a circle'''

import turtle

t = turtle.Turtle()

import math

'''Draws a circle replicating turtle's .cirlce() method.'''

def drawCircle(radius):

t.down()

distance = 2 \* math.pi \* radius / 120

for count in range(120):

t.forward(distance)

t.left(3)

return distance

'''Main function of the program'''

def main():

t.home

radius = int(input("Enter radius: "))

t.speed(1)

drawCircle(radius)

turtle.done()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Outputs:

1.

Enter radius: 500

Shape, circle

Description automatically generated

2.

Enter radius: 200

Shape, circle

Description automatically generated

3.

Enter radius: 50

A picture containing scatter chart

Description automatically generated

4.

Enter radius: 89

Shape

Description automatically generated with low confidence

Here is the video comparison:

>>> https://youtu.be/cb5yuSWcNbk

# Exercise 2

'''This function draws a "countryside church building minus the cross.", or as I like to call it, a crude picture of a house.'''

import turtle

t = turtle.Turtle()

t.up()

t.shape("turtle")

'''Main function of the program. Gathers input and then calls drawHouse()'''

def main():

scale = (input("Please enter the scale size between 0 and 5. Press enter for default value. "))

if scale == "" or scale == "/n":

scale = 1.0

else:

scale = float(scale)

if scale > 5:

print("Scale size needs to be between 0 and 5.")

main()

blc = float(input("Please enter an x coordinate. ")), float(input("Please enter a y coordinate. "))

drawHouse(blc, scale)

turtle.done()

'''Handles all of the functions required to draw the house'''

def drawHouse(blc, scale):

distance = 100 \* scale

t.goto(blc)

drawSquare(distance)

drawRoof(distance)

t.goto(blc)

drawWindow(distance)

t.goto(blc)

drawDoor(distance)

t.goto(blc)

'''Draws a square'''

def drawSquare(distance):

t.setheading(0)

t.down()

for i in range(4):

t.forward(distance)

t.left(90)

t.up()

'''Draws the roof of the house'''

def drawRoof(distance):

t.up()

t.setheading(90)

t.forward(distance)

t.setheading(60)

t.down()

for i in range(3):

t.forward(distance)

t.right(120)

t.setheading(0)

t.up()

'''Draws a window. Filled in 4 major quadrants to replicate the Microsoft Windows logo as a fun gag'''

def drawWindow(distance):

windis = distance / 6

t.setheading(0)

t.up()

t.forward(windis)

t.left(90)

t.forward(windis)

windis = windis / 2

t.fillcolor("blue")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.setheading(90)

t.forward(windis)

t.fillcolor("red")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.forward(windis)

t.fillcolor("green")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.setheading(270)

t.forward(windis)

t.fillcolor("yellow")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.up()

'''Draws the door and a handle and a circular knov'''

def drawDoor(distance):

t.setheading(0)

t.forward(distance \* 7/8)

t.setheading(90)

t.down()

for i in range(2):

t.forward(distance/2)

t.left(90)

t.forward(distance/4)

t.left(90)

t.up()

t.setheading(90)

t.forward(distance/4)

t.left(90)

t.forward(distance/20)

t.down()

t.circle(distance / 30)

t.up()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Outputs:

Note: I added fills for the window to replicate the windows logo. I did this to not only challenge myself but also to have fun with the project. The program still works as intended.

1.

Please enter the scale size between 0 and 5. Press enter for default value. 4

Please enter an x coordinate. -100

Please enter a y coordinate. -200

A picture containing diagram

Description automatically generated

2.

Please enter the scale size between 0 and 5. Press enter for default value.

Please enter an x coordinate. 89

Please enter a y coordinate. 124

A picture containing graphical user interface

Description automatically generated

3.

Please enter the scale size between 0 and 5. Press enter for default value. 2.8

Please enter an x coordinate. 300

Please enter a y coordinate. -100

Chart, line chart

Description automatically generated

4.

Please enter the scale size between 0 and 5. Press enter for default value. 5

Please enter an x coordinate. 0

Please enter a y coordinate. 0

Chart, box and whisker chart

Description automatically generated

# Exercise 3

'''This program draws a Christmas tree with ornaments'''

import turtle

import random

t = turtle.Turtle()

t.up()

t.shape("turtle")

t.speed(300)

'''Main function of the program'''

def main():

scale = (input("Please enter the scale size. Press enter for default value. "))

if scale == "" or scale == "/n":

scale = 1.0

else:

scale = float(scale)

blc = float(input("Please enter an x coordinate. ")), float(input("Please enter a y coordinate. "))

drawTree(scale, blc)

t.goto(blc)

drawTrunk(scale)

turtle.done()

'''Handles the fucntions required to draw the tree'''

def drawTree(scale, blc):

t.up()

t.goto(blc)

distance = scale \* 300

drawTriangle(distance)

drawOrn(blc, distance)

for i in range(2):

t.left(60)

t.forward(distance)

distance /= 2

t.setheading(270)

t.forward(distance / 2)

t.setheading(180)

t.forward(distance / 2)

drawTriangle(distance)

blc = t.position()

drawOrn(blc, distance)

'''Draws the general 3 triangles required to make up the tree'''

def drawTriangle(distance):

t.setheading(0)

t.down()

t.fillcolor("green")

t.begin\_fill()

for i in range(3):

t.forward(distance)

t.left(120)

t.end\_fill()

t.up()

'''Draws ornaments in a pattern with random colors'''

def drawOrn(blc, distance):

t.goto(blc)

t.setheading(30)

t.forward(distance \* 0.2)

t.setheading(0)

t.up()

for i in range(3):

drawCircleSet(distance)

t.left(120)

t.goto(blc)

t.setheading(0)

'''Used by drawOrn() to draw a "set" of cirlces. Just makes it easier to draw a set of ornaments per triangle'''

def drawCircleSet(distance):

for i in range(2):

color = pickColor()

t.forward(distance / 3)

t.down()

t.fillcolor(color)

t.begin\_fill()

t.circle(distance \* .04)

t.end\_fill()

t.up()

'''Picks a random color'''

def pickColor():

num = random.randint(1, 3)

if num == 1:

color = "blue"

elif num == 2:

color = "red"

elif num == 3:

color = "yellow"

return color

'''Draws the tree trunk'''

def drawTrunk(scale):

t.fillcolor("brown")

t.up()

t.forward(scale \* 300 \* 1/3)

distance = scale \* 100

t.down()

t.begin\_fill()

t.setheading(0)

for i in range(2):

t.forward(distance)

t.right(90)

t.forward(distance \* 0.8)

t.right(90)

t.end\_fill()

t.up()

if \_\_name\_\_ == "\_\_main\_\_":

main()

1.

Please enter the scale size. Press enter for default value.

Please enter an x coordinate. -100

Please enter a y coordinate. -100

Shape

Description automatically generated

2.

Please enter the scale size. Press enter for default value. 2

Please enter an x coordinate. -300

Please enter a y coordinate. -300

Shape

Description automatically generated

# Exercise 4

'''This program draws a snowman'''

import turtle

import math

import random

'''Main function of program. Gathers input and then calls drawSnowman()'''

def main():

t = turtle.Turtle()

turtle.tracer(False)

scale = input("Enter a scale or press enter for default value: ")

if scale == "" or scale == "/n":

scale = 1.0

else:

scale = float(scale)

blc = int(input("Enter x coordinate: ")), int(input("Enter y coordinate: "))

drawSnowman(t, scale, blc)

turtle.done()

'''Handles the functions for drawing the snowman'''

def drawSnowman(t, scale, blc):

t.up()

t.goto(blc)

radius = drawBody(t, scale)

drawFace(t, scale)

t.up()

t.goto(blc)

drawHat(t, scale, radius)

t.up()

t.goto(blc)

t.setheading(90)

t.forward(radius \* 2 \* 3/4)

blc = t.position()

drawArms(t, scale, blc, radius)

'''Draws the body using a for loop'''

def drawBody(t, scale):

radius = 40 \* scale

t.fillcolor("white")

for i in range(3):

t.begin\_fill()

t.down()

t.circle(radius)

t.end\_fill()

t.up()

t.setheading(90)

t.forward(radius \* 2 \* 3/4)

t.setheading(0)

t.up()

t.right(90)

t.forward(radius \* 2 \* 3/4)

return radius

'''Draws the face. I seperated it out a bit (not into fucntions, to avoid having a lot of them) into seperate sections for the different parts of the face. Those sections will be annotated accordingly'''

def drawFace(t, scale):

'''Draws the smile'''

t.up()

t.setheading(90)

t.forward(15 \* scale)

smilepos = t.position()

t.fillcolor("black")

for i in range (3):

t.setheading(0)

t.down()

t.begin\_fill()

t.circle(5 \* scale )

t.end\_fill()

t.up()

if i == 0:

t.left(20)

else:

t.left(20 \* i)

t.forward(13 \* scale)

t.goto(smilepos)

t.setheading(180)

t.right(40)

t.forward(16 \* scale)

for i in range (2):

t.setheading(180)

t.down()

t.begin\_fill()

t.circle(5 \* scale )

t.end\_fill()

t.up()

if i == 0:

t.right(30)

else:

t.right(40)

t.forward(13 \* scale)

'''Draws the nose'''

t.up()

t.goto(smilepos)

t.setheading(90)

t.forward(20 \* scale)

t.setheading(0)

t.left(15)

base = 10 \* scale

height = 30 \* scale

sidelen = math.sqrt(12 \*\* 2 / 4 + height \*\* 2)

t.down()

t.fillcolor("orange")

t.begin\_fill()

t.forward(sidelen)

t.left(150)

t.forward(sidelen)

t.left(105)

t.forward(base)

t.end\_fill()

'''Draws the eyes'''

t.fillcolor("black")

t.up()

t.goto(smilepos)

t.setheading(90)

t.forward(50 \* scale)

smilepos = t.position()

t.left(90)

t.forward(15 \* scale)

t.down()

t.begin\_fill()

t.circle(8 \* scale)

t.end\_fill()

t.up()

t.goto(smilepos)

t.setheading(0)

t.forward(15 \* scale)

t.begin\_fill()

t.down()

t.setheading(180)

t.circle(8 \* scale)

t.end\_fill()

t.up

'''Draws the arms, with the fingers being at random angles'''

def drawArms(t, scale, blc, radius):

t.width(5 \* scale)

t.pencolor("brown")

t.setheading(90)

t.up()

t.forward(radius)

t.right(90)

t.forward(radius)

t.left(30)

t.down()

arm = 40 \* scale

t.forward(arm \* 7/8)

hand = t.position()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.down()

t.setheading(30)

angle = random.randint(50, 150)

t.left(angle)

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(30)

angle = random.randint(50, 150)

t.right(angle)

t.down()

t.forward(arm \* 1/8)

t.up()

t.goto(blc)

t.setheading(90)

t.forward(radius)

t.left(90)

t.forward(radius)

t.setheading(150)

t.down()

t.forward(arm \* 7/8)

hand = t.position()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(150)

angle = random.randint(50, 150)

t.left(angle)

t.down()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(150)

angle = random.randint(50, 150)

t.down()

t.right(angle)

t.forward(arm \* 1/8)

t.up()

t.goto(blc)

t.hideturtle()

'''Draws a lil hat for him'''

def drawHat(t, scale, radius):

t.setheading(90)

for i in range(2):

t.forward(radius \* 2 \* 3/4)

t.forward(radius \* 2)

t.setheading(0)

t.down()

t.fillcolor("black")

t.begin\_fill()

t.forward(15 \* scale)

t.left(90)

t.forward(5 \* scale)

t.left(90)

t.forward(10 \* scale)

t.right(90)

t.forward(20 \* scale)

t.left(90)

t.forward(12 \* scale)

t.left(90)

t.forward(20 \* scale)

t.right(90)

t.forward(10 \* scale)

t.left(90)

t.forward(5 \* scale)

t.left(90)

t.forward(15 \* scale)

t.end\_fill()

t.up()

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 1.

# Enter a scale or press enter for default value:

# Enter x coordinate: -50

# Enter y coordinate: -50

# 

2.

Enter a scale or press enter for default value: 2

Enter x coordinate: -200

Enter y coordinate: -200

Diagram, shape

Description automatically generated with medium confidence

# Exercise 5

'''Draws a mystical, winter scene...'''

import turtle

import math

import random

t = turtle.Turtle()

t.up()

'''Main function of program, gathers input before passing to winterScene()'''

def main():

turtle.tracer(True)

scale = input("Enter a scale or press enter for default value: ")

if scale == "" or scale == "/n":

scale = 1.0

else:

scale = float(scale)

blc = int(input("Enter x coordinate: ")), int(input("Enter y coordinate: "))

winterScene(blc, scale)

turtle.done()

def winterScene(blc, scale):

housescale = scale \* 2

drawHouse(blc, housescale)

t.up()

t.goto(blc)

t.setheading(0)

t.forward(300 \* scale)

blc = t.position()

smscale = scale \* 1/2

drawSnowman(smscale, blc)

t.up()

t.goto(blc)

t.setheading(0)

t.forward(80 \* scale)

t.left(90)

t.forward(50 \* scale)

blc = t.position()

tscale = scale \* 0.8

t.width(1)

t.pencolor("black")

drawTree(tscale, blc)

t.goto(blc)

drawTrunk(tscale)

print("Walking in a winter wonderland...")

'''Handles all of the functions required to draw the house'''

def drawHouse(blc, scale):

distance = 100 \* scale

t.goto(blc)

drawSquare(distance)

drawRoof(distance)

t.goto(blc)

drawWindow(distance)

t.goto(blc)

drawDoor(distance)

t.goto(blc)

'''Draws a square'''

def drawSquare(distance):

t.setheading(0)

t.down()

for i in range(4):

t.forward(distance)

t.left(90)

t.up()

'''Draws the roof of the house'''

def drawRoof(distance):

t.up()

t.setheading(90)

t.forward(distance)

t.setheading(60)

t.down()

for i in range(3):

t.forward(distance)

t.right(120)

t.setheading(0)

t.up()

'''Draws a window. Filled in 4 major quadrants to replicate the Microsoft Windows logo as a fun gag'''

def drawWindow(distance):

windis = distance / 6

t.setheading(0)

t.up()

t.forward(windis)

t.left(90)

t.forward(windis)

windis = windis / 2

t.fillcolor("blue")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.setheading(90)

t.forward(windis)

t.fillcolor("red")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.forward(windis)

t.fillcolor("green")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.setheading(270)

t.forward(windis)

t.fillcolor("yellow")

t.begin\_fill()

drawSquare(windis)

t.end\_fill()

t.up()

'''Draws the door and a handle and a circular knov'''

def drawDoor(distance):

t.setheading(0)

t.forward(distance \* 7/8)

t.setheading(90)

t.down()

for i in range(2):

t.forward(distance/2)

t.left(90)

t.forward(distance/4)

t.left(90)

t.up()

t.setheading(90)

t.forward(distance/4)

t.left(90)

t.forward(distance/20)

t.down()

t.circle(distance / 30)

t.up()

'''Handles the functions for drawing the snowman'''

def drawSnowman(scale, blc):

t.up()

t.goto(blc)

radius = drawBody(scale)

drawFace(scale)

t.up()

t.goto(blc)

drawHat(scale, radius)

t.up()

t.goto(blc)

t.setheading(90)

t.forward(radius \* 2 \* 3/4)

blc = t.position()

drawArms(scale, blc, radius)

'''Draws the body using a for loop'''

def drawBody(scale):

radius = 40 \* scale

t.fillcolor("white")

for i in range(3):

t.begin\_fill()

t.down()

t.circle(radius)

t.end\_fill()

t.up()

t.setheading(90)

t.forward(radius \* 2 \* 3/4)

t.setheading(0)

t.up()

t.right(90)

t.forward(radius \* 2 \* 3/4)

return radius

'''Draws the face. I seperated it out a bit (not into fucntions, to avoid having a lot of them) into seperate sections for the different parts of the face. Those sections will be annotated accordingly'''

def drawFace(scale):

'''Draws the smile'''

t.up()

t.setheading(90)

t.forward(15 \* scale)

smilepos = t.position()

t.fillcolor("black")

for i in range (3):

t.setheading(0)

t.down()

t.begin\_fill()

t.circle(5 \* scale )

t.end\_fill()

t.up()

if i == 0:

t.left(20)

else:

t.left(20 \* i)

t.forward(13 \* scale)

t.goto(smilepos)

t.setheading(180)

t.right(40)

t.forward(16 \* scale)

for i in range (2):

t.setheading(180)

t.down()

t.begin\_fill()

t.circle(5 \* scale )

t.end\_fill()

t.up()

if i == 0:

t.right(30)

else:

t.right(40)

t.forward(13 \* scale)

'''Draws the nose'''

t.up()

t.goto(smilepos)

t.setheading(90)

t.forward(20 \* scale)

t.setheading(0)

t.left(15)

base = 10 \* scale

height = 30 \* scale

sidelen = math.sqrt(12 \*\* 2 / 4 + height \*\* 2)

t.down()

t.fillcolor("orange")

t.begin\_fill()

t.forward(sidelen)

t.left(150)

t.forward(sidelen)

t.left(105)

t.forward(base)

t.end\_fill()

'''Draws the eyes'''

t.fillcolor("black")

t.up()

t.goto(smilepos)

t.setheading(90)

t.forward(50 \* scale)

smilepos = t.position()

t.left(90)

t.forward(15 \* scale)

t.down()

t.begin\_fill()

t.circle(8 \* scale)

t.end\_fill()

t.up()

t.goto(smilepos)

t.setheading(0)

t.forward(15 \* scale)

t.begin\_fill()

t.down()

t.setheading(180)

t.circle(8 \* scale)

t.end\_fill()

t.up

'''Draws the arms, with the fingers being at random angles'''

def drawArms(scale, blc, radius):

t.width(5 \* scale)

t.pencolor("brown")

t.setheading(90)

t.up()

t.forward(radius)

t.right(90)

t.forward(radius)

t.left(30)

t.down()

arm = 40 \* scale

t.forward(arm \* 7/8)

hand = t.position()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.down()

t.setheading(30)

angle = random.randint(50, 150)

t.left(angle)

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(30)

angle = random.randint(50, 150)

t.right(angle)

t.down()

t.forward(arm \* 1/8)

t.up()

t.goto(blc)

t.setheading(90)

t.forward(radius)

t.left(90)

t.forward(radius)

t.setheading(150)

t.down()

t.forward(arm \* 7/8)

hand = t.position()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(150)

angle = random.randint(50, 150)

t.left(angle)

t.down()

t.forward(arm \* 1/8)

t.up()

t.goto(hand)

t.setheading(150)

angle = random.randint(50, 150)

t.down()

t.right(angle)

t.forward(arm \* 1/8)

t.up()

t.goto(blc)

'''Draws a lil hat for him'''

def drawHat(scale, radius):

t.setheading(90)

for i in range(2):

t.forward(radius \* 2 \* 3/4)

t.forward(radius \* 2)

t.setheading(0)

t.down()

t.fillcolor("black")

t.begin\_fill()

t.forward(15 \* scale)

t.left(90)

t.forward(5 \* scale)

t.left(90)

t.forward(10 \* scale)

t.right(90)

t.forward(20 \* scale)

t.left(90)

t.forward(12 \* scale)

t.left(90)

t.forward(20 \* scale)

t.right(90)

t.forward(10 \* scale)

t.left(90)

t.forward(5 \* scale)

t.left(90)

t.forward(15 \* scale)

t.end\_fill()

t.up()

'''Handles the fucntions required to draw the tree'''

def drawTree(scale, blc):

t.up()

t.goto(blc)

distance = scale \* 300

drawTriangle(distance)

drawOrn(blc, distance)

for i in range(2):

t.left(60)

t.forward(distance)

distance /= 2

t.setheading(270)

t.forward(distance / 2)

t.setheading(180)

t.forward(distance / 2)

drawTriangle(distance)

blc = t.position()

drawOrn(blc, distance)

'''Draws the general 3 triangles required to make up the tree'''

def drawTriangle(distance):

t.setheading(0)

t.down()

t.fillcolor("green")

t.begin\_fill()

for i in range(3):

t.forward(distance)

t.left(120)

t.end\_fill()

t.up()

'''Draws ornaments in a pattern with random colors'''

def drawOrn(blc, distance):

t.goto(blc)

t.setheading(30)

t.forward(distance \* 0.2)

t.setheading(0)

t.up()

for i in range(3):

drawCircleSet(distance)

t.left(120)

t.goto(blc)

t.setheading(0)

'''Used by drawOrn() to draw a "set" of cirlces. Just makes it easier to draw a set of ornaments per triangle'''

def drawCircleSet(distance):

for i in range(2):

color = pickColor()

t.forward(distance / 3)

t.down()

t.fillcolor(color)

t.begin\_fill()

t.circle(distance \* .04)

t.end\_fill()

t.up()

'''Picks a random color'''

def pickColor():

num = random.randint(1, 3)

if num == 1:

color = "blue"

elif num == 2:

color = "red"

elif num == 3:

color = "yellow"

return color

'''Draws the tree trunk'''

def drawTrunk(scale):

t.fillcolor("brown")

t.up()

t.forward(scale \* 300 \* 1/3)

distance = scale \* 100

t.down()

t.begin\_fill()

t.setheading(0)

for i in range(2):

t.forward(distance)

t.right(90)

t.forward(distance \* 0.8)

t.right(90)

t.end\_fill()

t.up()

if \_\_name\_\_ == "\_\_main\_\_":

main()

1.

Enter a scale or press enter for default value:

Enter x coordinate: -300

Enter y coordinate: -300

Walking in a winter wonderland...

Shape

Description automatically generated

2.

Enter a scale or press enter for default value: 2

Enter x coordinate: -400

Enter y coordinate: -400

Walking in a winter wonderland...

Shape, square

Description automatically generated