Python Turtle Graphics 2 – Longlevens Code Club.

Version 2.1 191130

Hello.

Welcome to part 2 of the Turtle graphics tutorial for python.

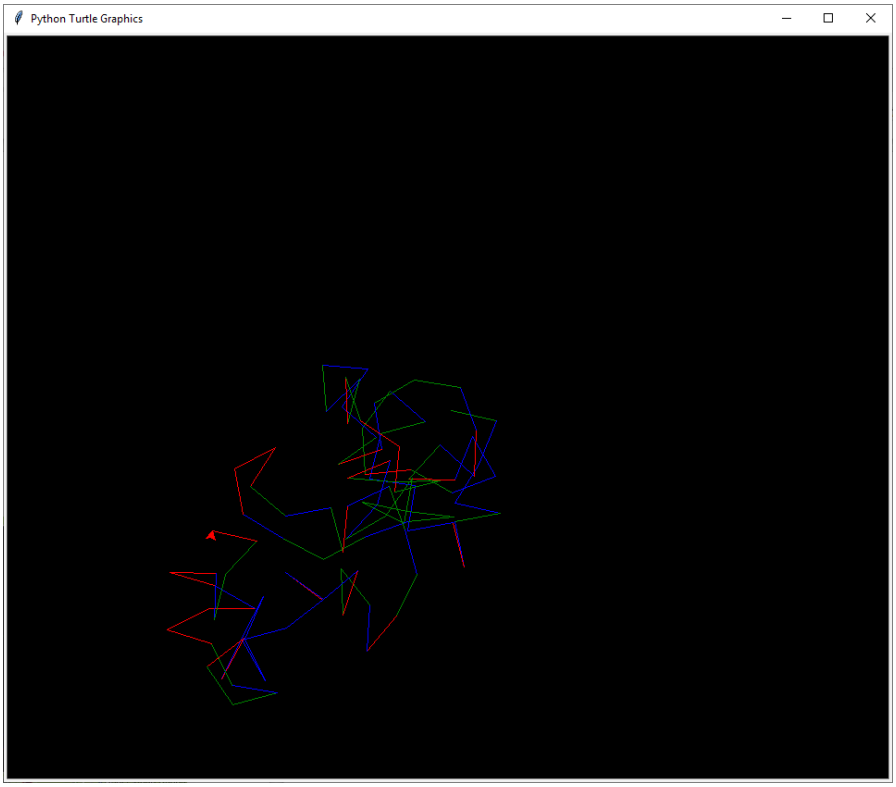
As you are now getting better in writing python programs, Here are a few rules you will need to follow.

- Give each program to easy to remember name
- Remember to save your program often.
- Indents are important
- If you repeat the same code to do something – should it be a function?
- # is used to add comments so EVERYONE can understand what you are doing
- use **t.screen.exitonclick()** if your program ends before you want it to.

In this part we are going to learn:

1. How to react to mouse presses and mouse movement
2. How to process keyboard presses
3. To practice creating functions
4. To create some great animations
5. To write better programs

Remember the random walking turtle from the previous tutorial?



We're going to make it SUPER fast

t.speed() accepts an integer as an argument to set the speed of the turtle. The value 0(zero) is the fastest speed of the turtle

```
from turtle import Turtle, Screen
import random

t=Turtle()
window = Screen()
window.title("Random walk")

t.ht()
window.bgcolor("black")
#set turtle speed to maximum
t.speed(0)

def random_drawing(turns,distance):
        for x in range(turns):
                right=t.right(random.randint(0,360))
                left=t.left(random.randint(0,360))

                t.color(random.choice(["blue","red","green"]))
                random.choice([right,left])
                t.fd(distance)

random_drawing(1000,10)
window.exitonclick()
```

Finally, we can go even faster by turning off the drawing to the screen until we want to update it.

```
from turtle import Turtle, Screen
import random

t=Turtle()
window = Screen()
window.title("Random walk")

t.ht()
window.bgcolor("black")

#Turn off screen updates
window.tracer(0,0)

def random_drawing(turns,distance):
    for x in range(turns):
        right=t.right(random.randint(0,360))
        left=t.left(random.randint(0,360))

        t.color(random.choice(["blue","red","green"]))
        random.choice([right,left])
        t.fd(distance)


random_drawing(1000,10)
window.exitonclick()
```

Next we are going to look at using your keyboard and mouse to control your programs

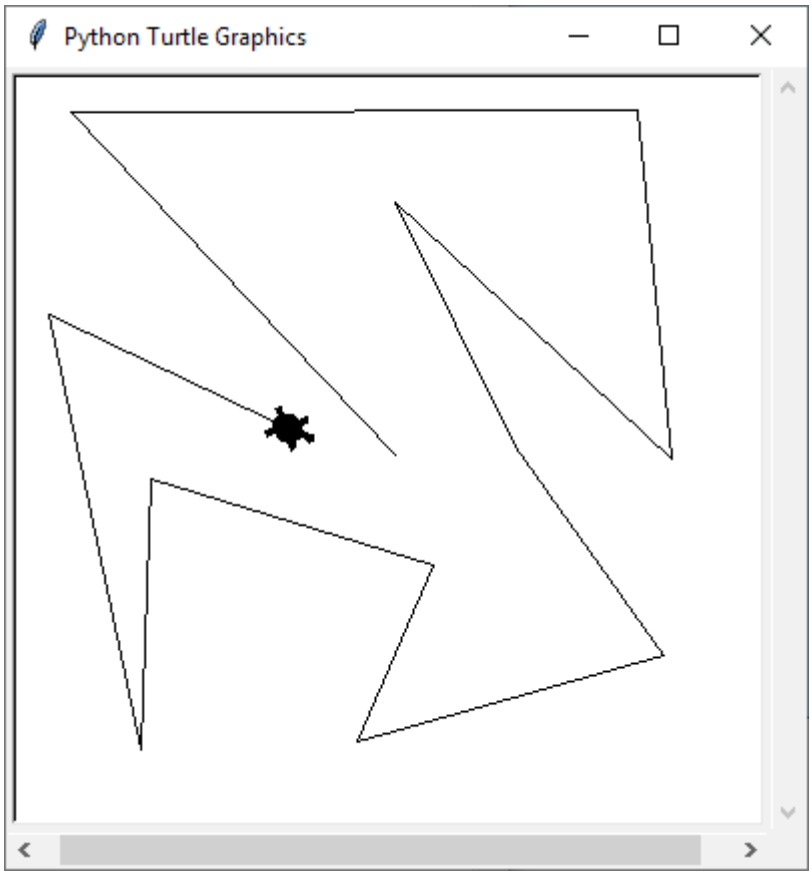
USING THE MOUSE

Let's write a program that shows you how the mouse position works..

```
from turtle import Screen

screen = Screen()
screen.setup(400,400)

#function called when mouse moved
def motion(event):
    x, y = event.x, event.y
    print('Motion function: {}, {}'.format(x, y))

#code to capture mouse motion anywhere in the window we've created
canvas = screen.getcanvas()
canvas.bind('<Motion>', motion)

screen.mainloop()
```

Now we want to detect when the mouse is clicked.

```
from turtle import Turtle, Screen

#setup screen

screen = Screen()
screen.setup(400,400)

t = Turtle('turtle')
t.speed('fastest')


#function called when mouse moved
def motion(event):
    x, y = event.x, event.y
    print('Motion function: {}, {}'.format(x, y))

def moveto(x,y):
    print("turtle clicked x:", x, " y:", y)
    t.onclick(None)
    t.setheading(t.towards(x, y))
    t.goto(x, y)
    t.onclick(moveto)

def get_mouse_click_coor(x, y):
    print("mouse clicked x:", x, " y:", y)
    t.setheading(t.towards(x, y))
    t.goto(x, y)



#attach mouse events to the screen

screen.onscreenclick(get_mouse_click_coor)

#code to capture mouse motion anywhere in the screen
canvas = t.screen.getcanvas()
canvas.bind('<Motion>', motion)

#loop forever
screen.mainloop()
```



Look at the new code. Notice the onclick() function

onclick()

This method calls a function whenever the mouse is used to click on a particular

coordinate. It accepts three arguments which are:

- ***function***– this is a function with two arguments, to which will be assigned the coordinates(x,y) of the clicked point on the canvas or **none**
- ***btn***– this argument is assigned to one by default, which means you click the left mouse button to call a function. If you assign it to two, you will have to click both the left and right mouse button simultaneously to call a function.
- ***add***– True or False. If True, a new binding will be added, otherwise, it will replace a former

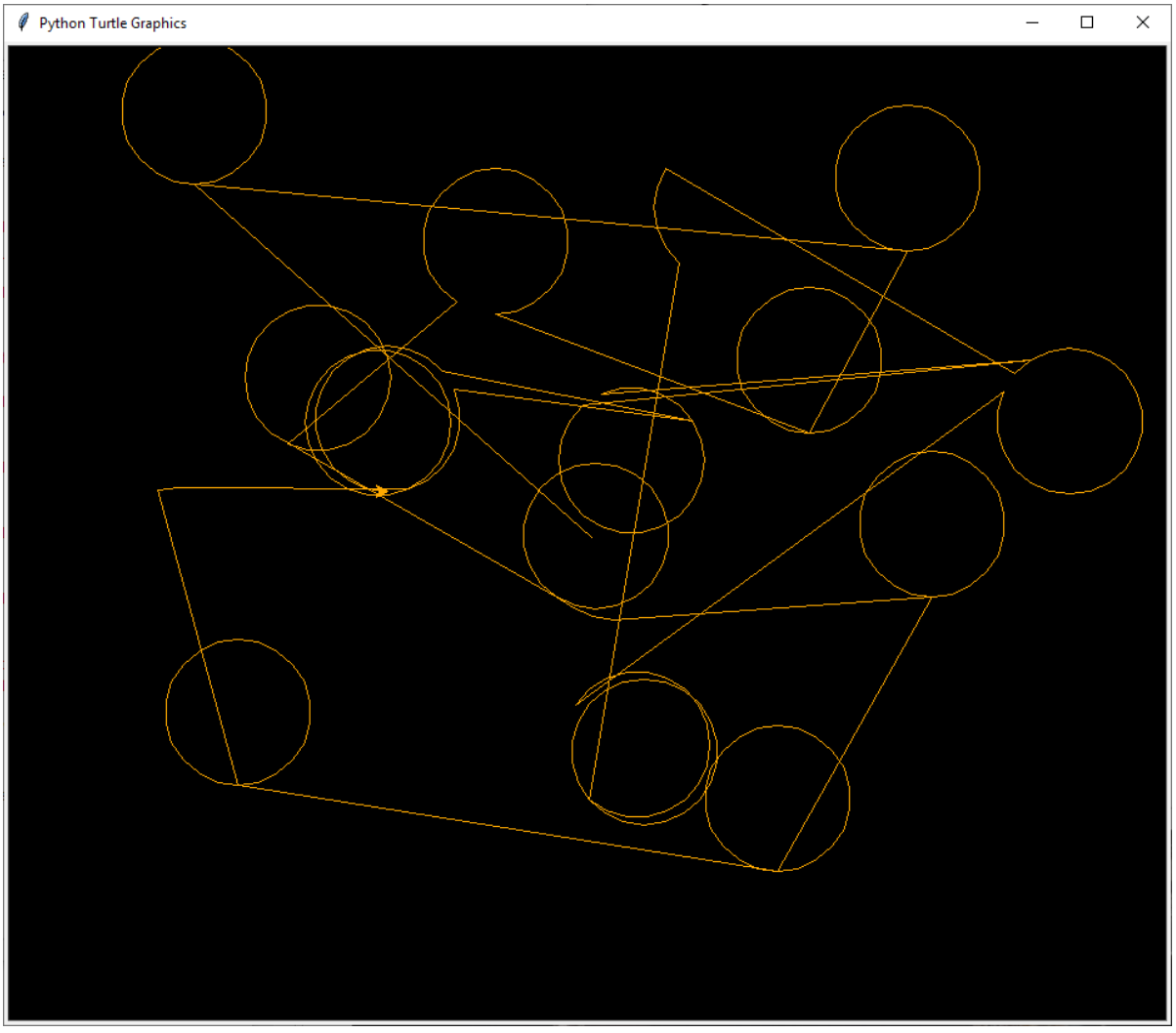
Now we'll get the turtle to do something else when we click on the screen

Here is a program that draws a circle when you click on the turtle.

Draw a circle with a mouse click

```
from turtle import Turtle, Screen

window = Screen()

t=Turtle()
window.bgcolor("black")
t.color("orange")

def mycircle(x,y):
  t.goto(x,y)
  t.circle(60)

window.onclick(mycircle)

window.mainloop()
```

The function name “mycircle” has two arguments, x and y which are the coordinates you will have in able to draw a circle.



ondrag() method

This method is basically used when you what to click on the turtle and move it.
It accepts three arguments like the onclick() method.

Here is a program to illustrate the ondrag() method.

```
from turtle import Turtle, Screen

#setup screen
screen = Screen()
screen.setup(400,400)

t = Turtle('turtle')


#function called when mouse moved
def motion(event):
    x, y = event.x, event.y
    print('Motion function: {}, {}'.format(x, y))

#function called when turtle is dragged
def dragging(x, y):
    t.ondrag(None)
    print("turtle dragged x:", x, " y:", y)
    t.setheading(t.towards(x, y))
    t.goto(x, y)
    t.ondrag(dragging)

def moveto(x,y):
    print("turtle clicked x:", x, " y:", y)
    t.onclick(None)
    t.setheading(t.towards(x, y))
    t.goto(x, y)
    t.onclick(moveto)

def get_mouse_click_coor(x, y):
    print("mouse clicked x:", x, " y:", y)
    t.setheading(t.towards(x, y))
    t.goto(x, y)


t.speed('fastest')

#attach mouse events to the turtle
t.ondrag(dragging)
t.onclick(moveto)
t.screen.onscreenclick(get_mouse_click_coor)

#code to capture mouse motion anywhere in the screen
canvas = t.screen.getcanvas()
canvas.bind('<Motion>', motion)

screen.mainloop()
```

onrelease() method

This method is also similar to onclick and ondrag method. To understand what this method does,

I wrote a program to draw a circle when you left click on the turtle and when you remove your hand from the mouse, the circle will disappear.

```
from turtle import Turtle, Screen

#setup screen
screen = Screen()
screen.setup(400,400)

t = Turtle('turtle')


#function called when mouse moved
def motion(event):
  x, y = event.x, event.y
  print('Motion function: {}, {}'.format(x, y))

#function called when turtle is dragged
def dragging(x, y):
  t.ondrag(None)
  print("turtle dragged x:", x, " y:", y)
  t.setheading(t.towards(x, y))
  t.goto(x, y)
  t.ondrag(dragging)

def moveto(x,y):
  print("turtle clicked x:", x, " y:", y)
  t.onclick(None)
  t.setheading(t.towards(x, y))
  t.goto(x, y)
  t.onclick(moveto)

def get_mouse_click_coor(x, y):
  print("mouse clicked x:", x, " y:", y)
  t.setheading(t.towards(x, y))
  t.goto(x, y)

def erase_drawing(x,y):
  t.clear()


t.speed('fastest')

#attach mouse events to the turtle
t.ondrag(dragging)
t.onclick(moveto)
t.onrelease(erase_drawing)
t.screen.onscreenclick(get_mouse_click_coor)


#code to capture mouse motion anywhere in the screen
canvas = t.screen.getcanvas()
canvas.bind('<Motion>', motion)

screen.mainloop()
```

How to control the turtle with your keyboard

This section is going to be fun as we will control the turtle with the keyboard of our

computer. Before we write some codes, let me explain the following concepts.

1. onkey()
2. listen()

onkey()

This method is similar to onclick() method. The only differences is that the onclick() method responds to clicks on the mouse while the onkey() method responds to commands from your computer keyboard.

The onkey() method accepts two arguments which are:

- ***fun:*** this is a function without any argument
- ***keyboard command:*** this is basically any alphabet on your keyboard (e.g n) or any keyboard command key. e.g space, your direction keys(up, down). You have to pass this command as a string

listen()

This method tells your program to listen to the keyboard.

Lets write a program to control the turtle to move up, down, left and right using the computer keyboard. The turtle should move constantly to any pixels of your choice.

Include an undo action.

Open a new file editor and save the following code snippet as **turtle_keyboard_control.py**

keyboard control program

```
from turtle import Turtle
t=Turtle()

#functions
def up():
    if not(t.heading() == 90):
        t.setheading(90)
        t.fd(50)
    else:
        t.fd(50)

def down():
    if not(t.heading() == 270):
        t.setheading(270)
        t.fd(50)
    else:
        t.fd(50)

def right():
    if not (t.heading() == 0):
        t.setheading(0)
        t.fd(50)
    else:
        t.fd(50)

def left():
    if not (t.heading() ==180):
        t.setheading(180)
        t.fd(50)
    else:
        t.fd(50)

def undo_button():
    t.undo()

#function where you add keyboard commands
def keyboard_commands():
    t.screen.onkey(up,"Up")
    t.screen.onkey(down,"Down")
    t.screen.onkey(right,"Right")
    t.screen.onkey(left,"Left")
    t.screen.onkey(undo_button,"End")
    t.screen.listen()

#main program
keyboard_commands()
```

How to Determine the Position of the Turtle.

position() method:

This method returns the position of the turtle, both the coordinates of the x and y axis is returned.

xcor() method:

This method returns the x coordinate of the turtle.

ycor() method:

This method returns the y coordinate of the turtle.

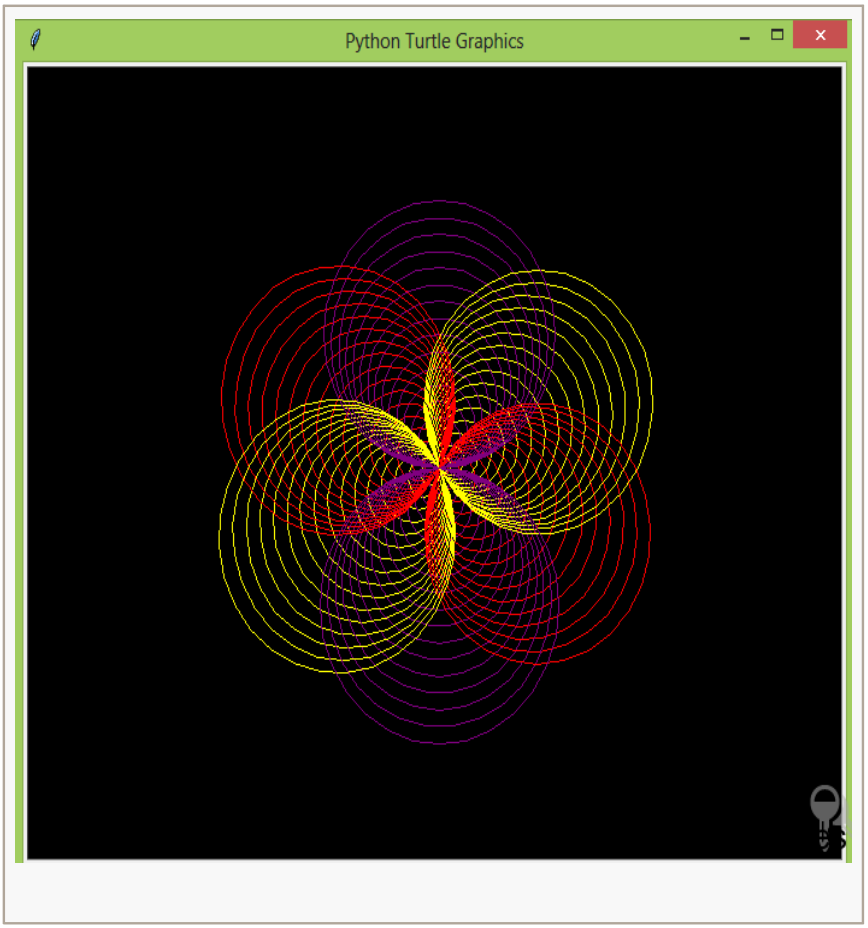
Can you add a function to the last program that prints the turtle's position when you press a key?

Ok.

Finally, we are going to write a few programs that will draw some creative shapes.

```
from turtle import Turtle
t=Turtle()

t.screen.bgcolor("black")
colours=["red","yellow","purple"]

for x in range(100):
  t.circle(x)
  t.color(colours[x%3])
  t.left(60)

t.screen.exitonclick()
```

Here’s the output of the above code snippet.



It's a bit slow isn't it.. So let's see if we can speed things up a bit using the speed(number) method

Add

t.speed(0) to your code

(the 0 means as fast as possible)

```
from turtle import Turtle

t=Turtle()

#run turtle as fast as it can
t.speed(0)

t.screen.bgcolor("black")
colours=["red","yellow","purple"]


for x in range(100):
  t.circle(x)
  t.color(colours[x%3])
  t.left(60)

t.screen.exitonclick()
```

It's an improvement but still not fast enough.

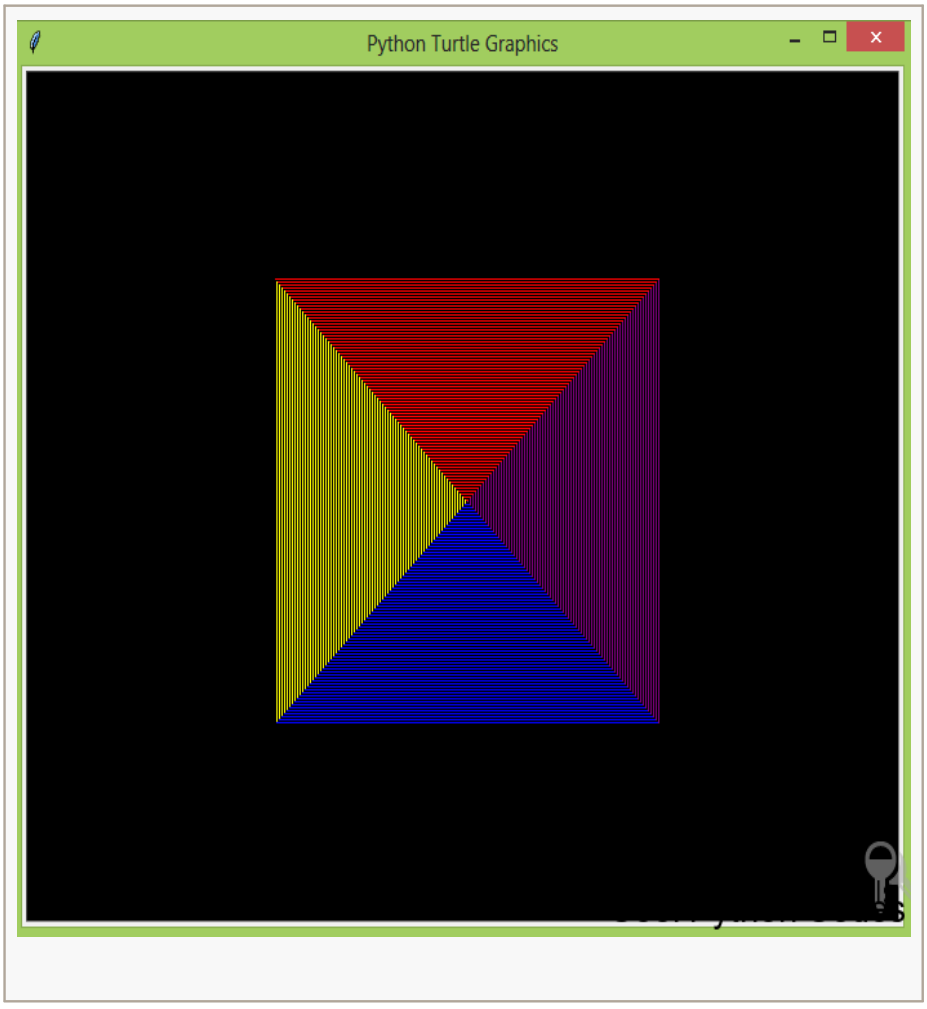
We can speed things up by only showing the graphics once everything has finished drawing.

```
from turtle import Turtle

t=Turtle()

# turn off showing graphics until all drawn
t.screen.tracer(0,0)

t.screen.bgcolor("black")
colours=["red","yellow","purple"]

for x in range(100):
  t.circle(x)
  t.color(colours[x%3])
  t.left(60)

#wait until we click to exit
t.screen.exitonclick()
```

See if you can add another colour

Let’s do a similar program with squares

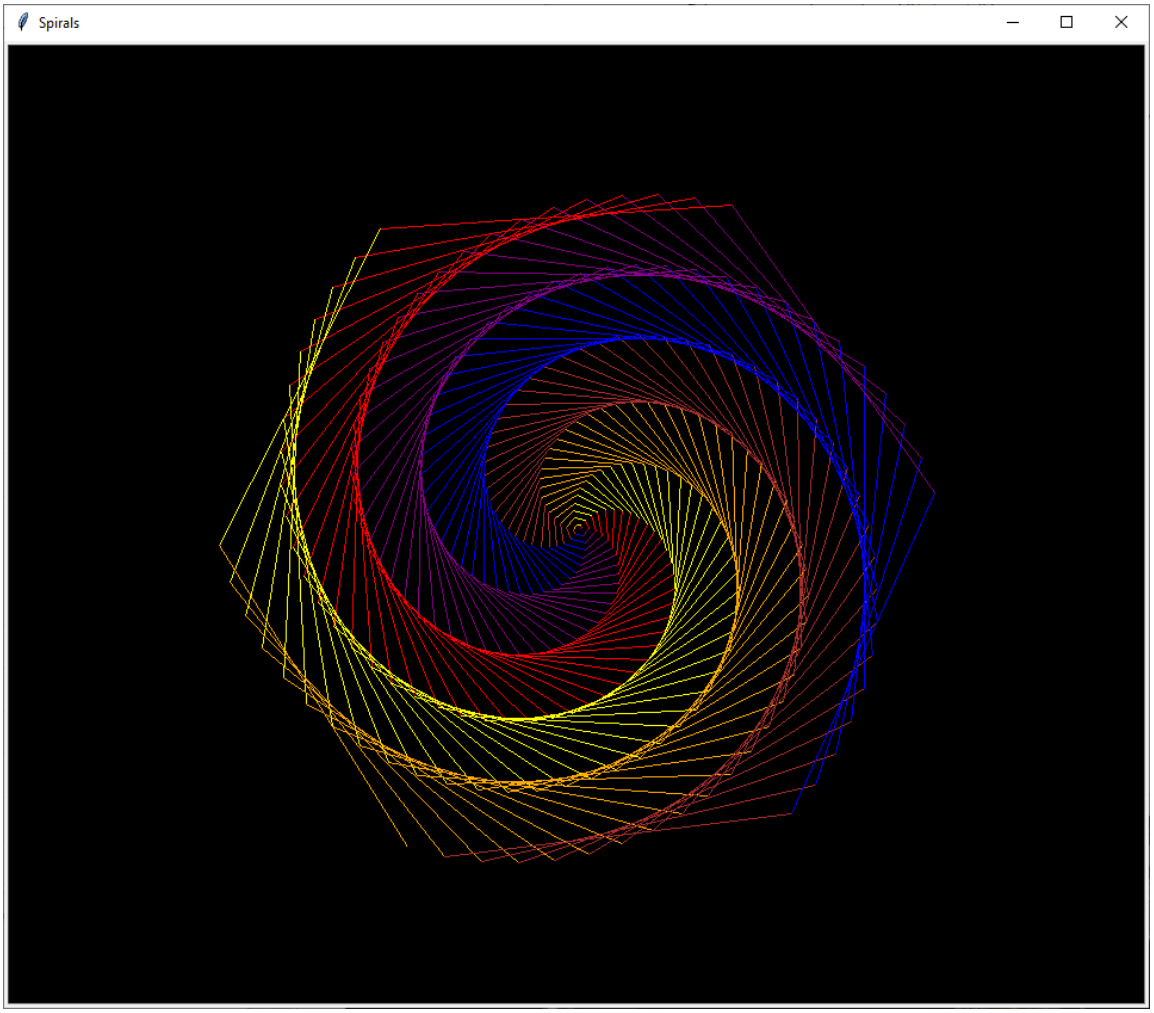
```
from turtle import Turtle

t=Turtle()
t.screen.bgcolor("black")
colours=["blue","purple","red","yellow"]
t.screen.tracer(0,0)

for x in range(300):
      t.color(colours[x%4])
      t.fd(x)
      t.left(90)

t.screen.exitonclick()
```



Ok. Let’s finish with a spiral

```
from turtle import Turtle

t=Turtle()
t.screen.bgcolor("black")

#Add a window title below

colours=["blue","purple","red","yellow","orange","brown"]
t.screen.tracer(0,0)

for x in range(300):
  t.color(colours[x%6])
  t.fd(x)
  t.left(59)

t.screen.exitonclick()
```

Here’s the output of the above code snippet.



Change the program to look like this below and see what it does

```
from turtle import Turtle

t=Turtle()
t.screen.bgcolor("black")

#Add a window title below

colours=["blue","purple","red","yellow","orange","brown"]
t.screen.tracer(0,0)

for x in range(300):
  t.color(colours[x%6])
  t.fd(x)

 #we've only changed the angle from 59 to 60.
  t.left(60)

t.screen.exitonclick()
```

Can you add another colour or draw a spiral wherever you click the mouse?

Congratulations - you get a star!!