Python Turtle Graphics – The Complete Guide.

How to Create a Turtle Window.

To create a turtle window on your computer screen, please open a new file editor window and type the following code snippet. Save it as **window1.py**

```
from turtle import Turtle
t=Turtle()
t.screen.exitonclick()
```

When you run this program, you will get this result on your screen.

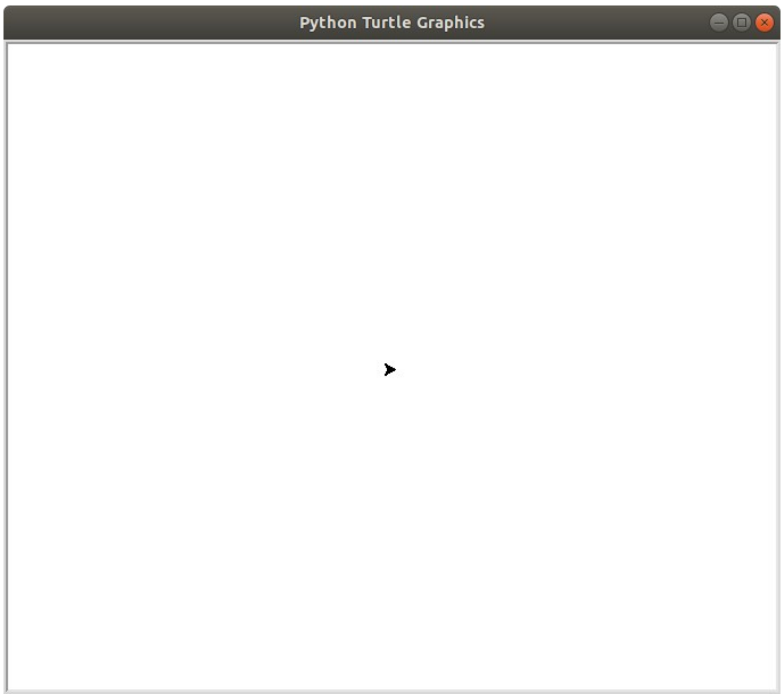
You can see an arrow head at the middle point of the window pointing to the east which is the default position. This location is called home.

The position of the turtle is specified with the two-dimensional coordinate system,
(x,y).

The default position of the turtle(arrow head) is (0,0) which is the center of the
window.

To make the turtle window disappear after viewing the turtle screen when you click on the turtle canvas, use the method **screen.exitonclick()** .

When you run the code snippet above the turtle screen won't disappear immediately unless you click on the turtle canvas.



How to Change the Turtle Window Background Color

The default background color is white which can be boring. You can change it using the method screen.bgcolor()

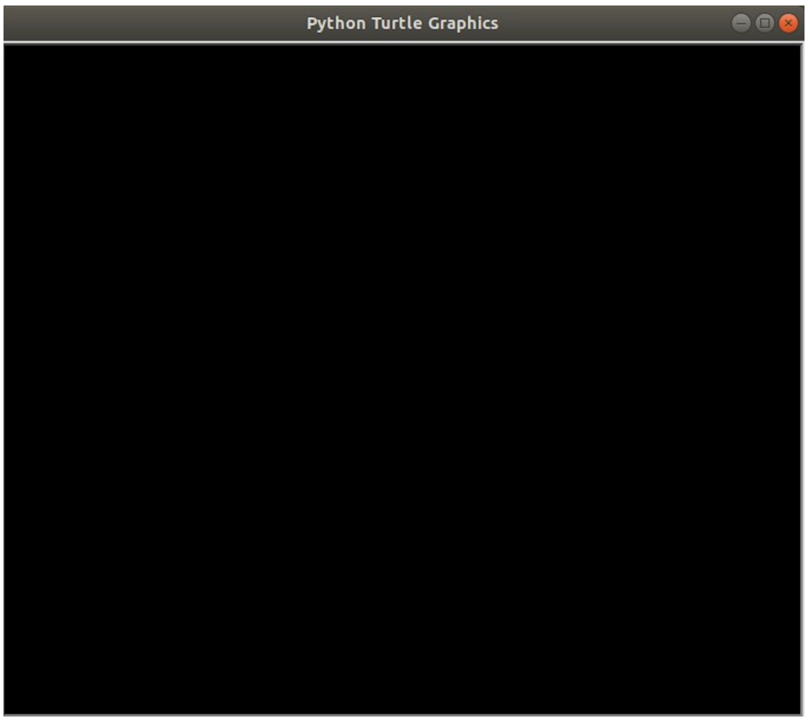
To change the color of the turtle window background, please open a new file editor window and type the following code snippet. Save it as **window2.py**

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor('black')
t.screen.exitonclick()
```

The ***screen.bgcolor()*** method accepts one argument which is the color of the turtle window as a string. The default color of Python turtle window is white.

When you run this program, you will get this result on your screen.

The background color of the turtle window is black but there is no turtle because the default color of the turtle is black.



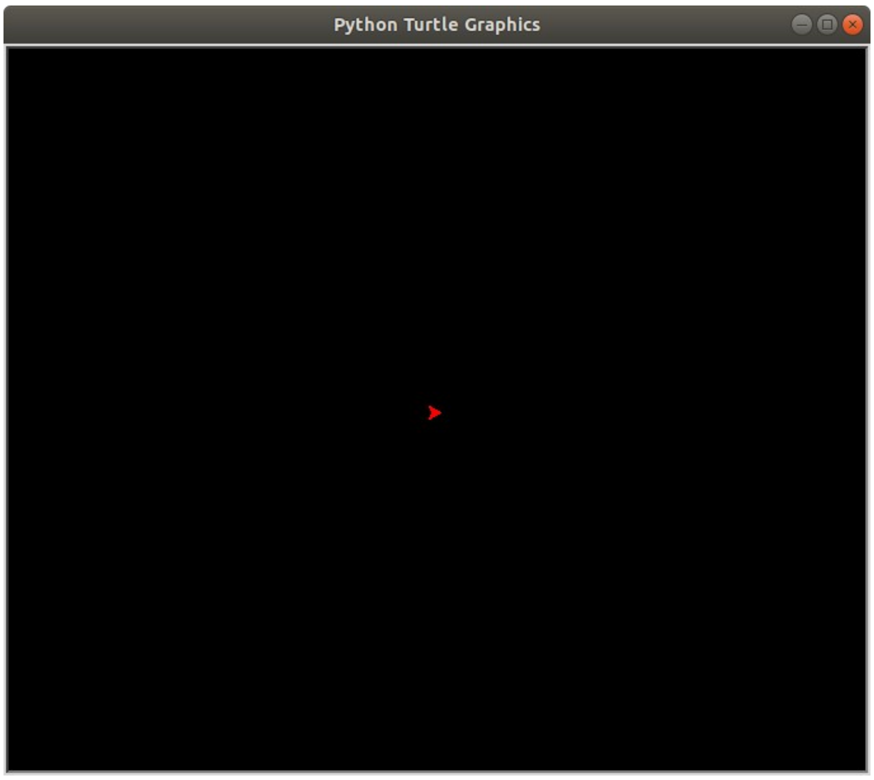
How to Change the Color of the Turtle

To change the color of the turtle,

Type the following code snippet into a new file editor and save it as any name you like.

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor('black')
t.color('red')

t.screen.exitonclick()
```

When you run this program, you will get this result on your screen.

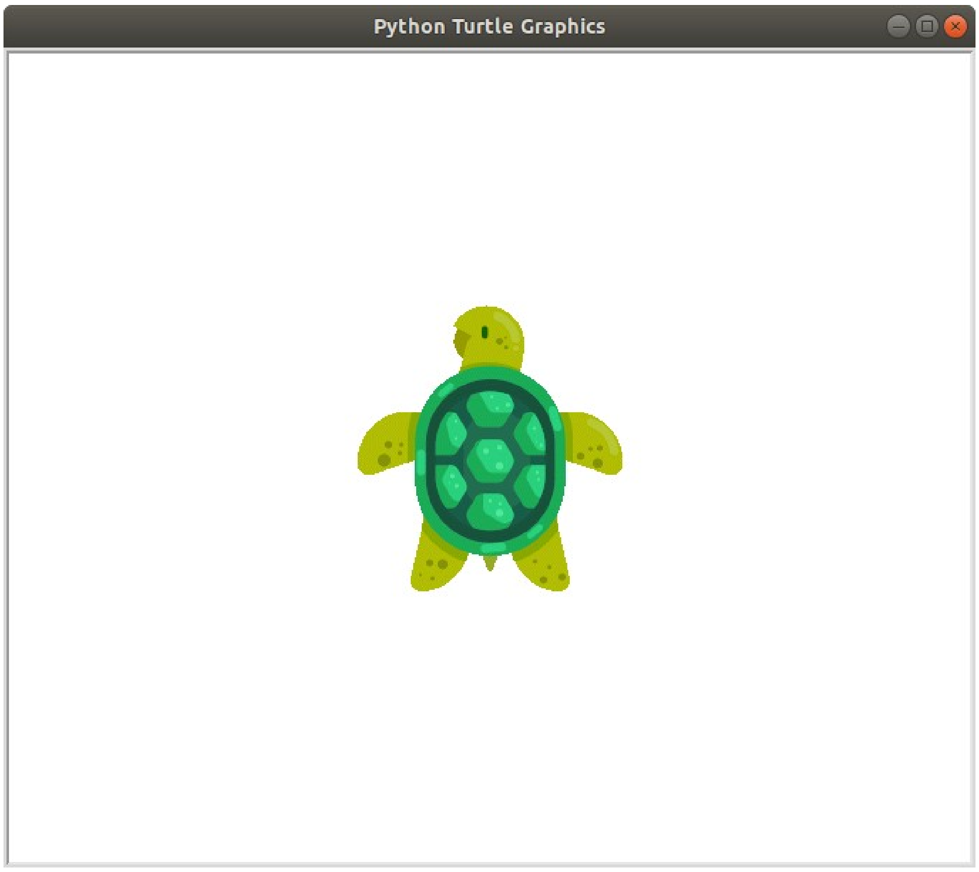


How to change the Background picture of the Turtle Screen.

Here’s a code snippet that changes the background image of the turtle screen (***turtle_pic.py***) .
How to change the turtle background image

```
from turtle import Turtle

t=Turtle()
t.hideturtle()
t.screen.bgpic("turtle.gif")
t.screen.exitonclick()
```

Here’s the output of the program (***turtle_pic.py***)



How to Set the Heading of the Turtle

•

By default the turtle faces the east, to change the direction in which the turtle is facing

```
from turtle import Turtle

t=Turtle()
t.hideturtle()
t.screen.bgpic("turtle.gif")
t.setheading(180)
t.screen.exitonclick()
```

The syntax is,

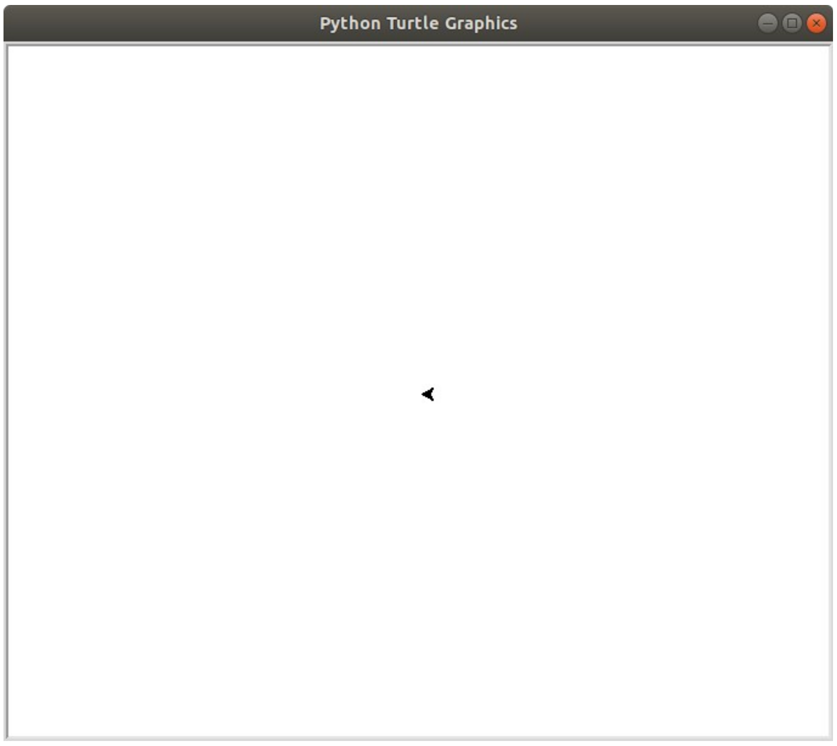
t.setheading(angle)

Angle 90 sets the turtle to face the North. Angle 180 sets the turtle to face the West. Angle 270 sets the turtle to face the South. Angle 360 or 0 set turtle to face the East
You can set the turtle to face any angle of your choice.

You can change the direction of the turtle using the following syntax

t.left(angle)

t.right(angle)



How to Change the Turtle Shape

The default shape of a turtle is known as a classic.

To change the shape of the turtle you use the ***shape()*** method.

The shape() method accepts a string as an argument. The string is the name of a valid shape in the TurtleScreen's shape dictionary.

The valid turtle shapes are:

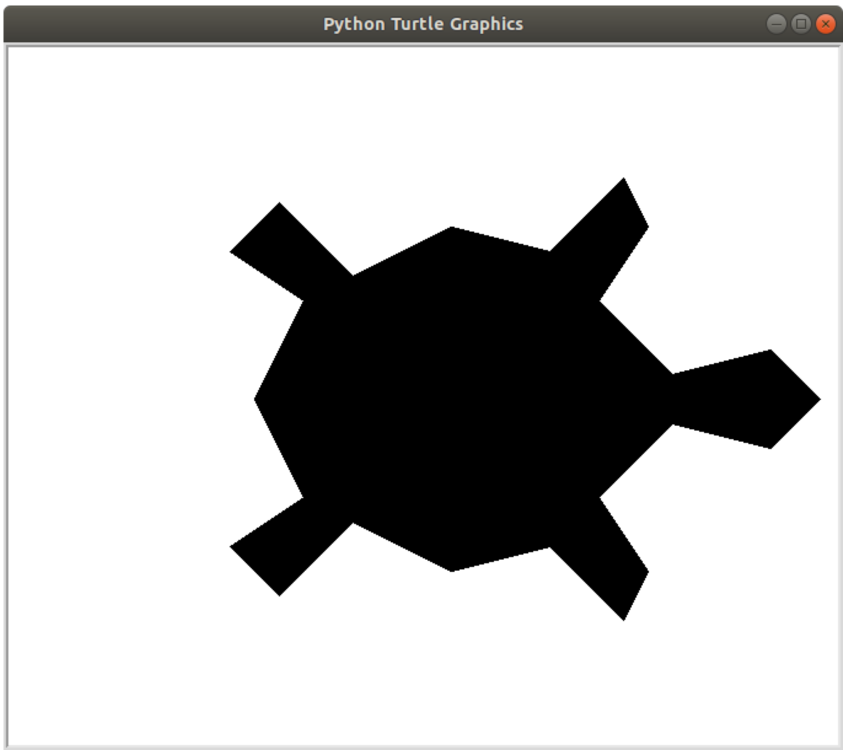
Arrow Circle
Square
Triangle
Turtle Classic

Open a new file editor window and type the following code snippet. Save it as **turtle_shape.py**

```
from turtle import Turtle
t=Turtle()
t.shape("turtle")
t.screen.exitonclick()
```

Change the turtle to the other types to see them

From the above screenshot, the turtle is small. To increase the size of the turtle, use the ***shapesize()*** method.
Add
t.shapesize(20)
Change the shapesize to different numbers to see how if looks.



How to Draw a Line

To draw a line use the syntax below

t.fd(distance)

or

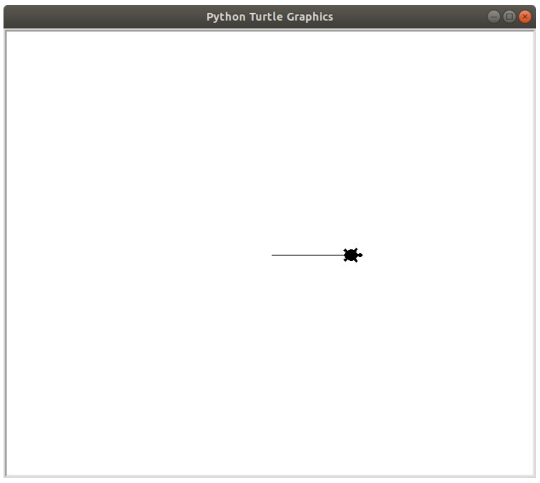
t.goto(x,y)

distance is the value of the length of the line.
t.goto(x,y) moves the turtle to an absolute
position **Note:**
The Cartesian system is the coordinate system for Turtle. So x and y represent values in the x and y axis respectively.

t.fd(distance) moves the turtle by the specified distance.

Open a new file editor window and type the following code snippet. Save it as **draw_line.py**

```
from turtle import Turtle
t=Turtle()
t.shape(“turtle”)
t.fd(100)
t.screen.exitonclick()
```

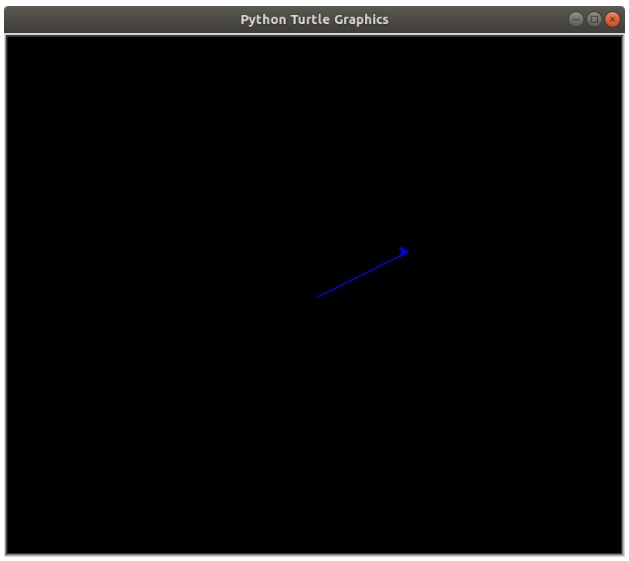


When you run the program above you will get:

The above program moves the turtle 100 pixels
to the east.

Open a new file editor window and type the following code snippet.
Save it as **turtle_go_to.py**

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

t.shape(“turtle”)
t.fd(100)
t.screen.exitonclick()
```



And then change to to following.

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

t.screen.bgcolor("black")
t.color("blue")
t.goto(100,50)
```

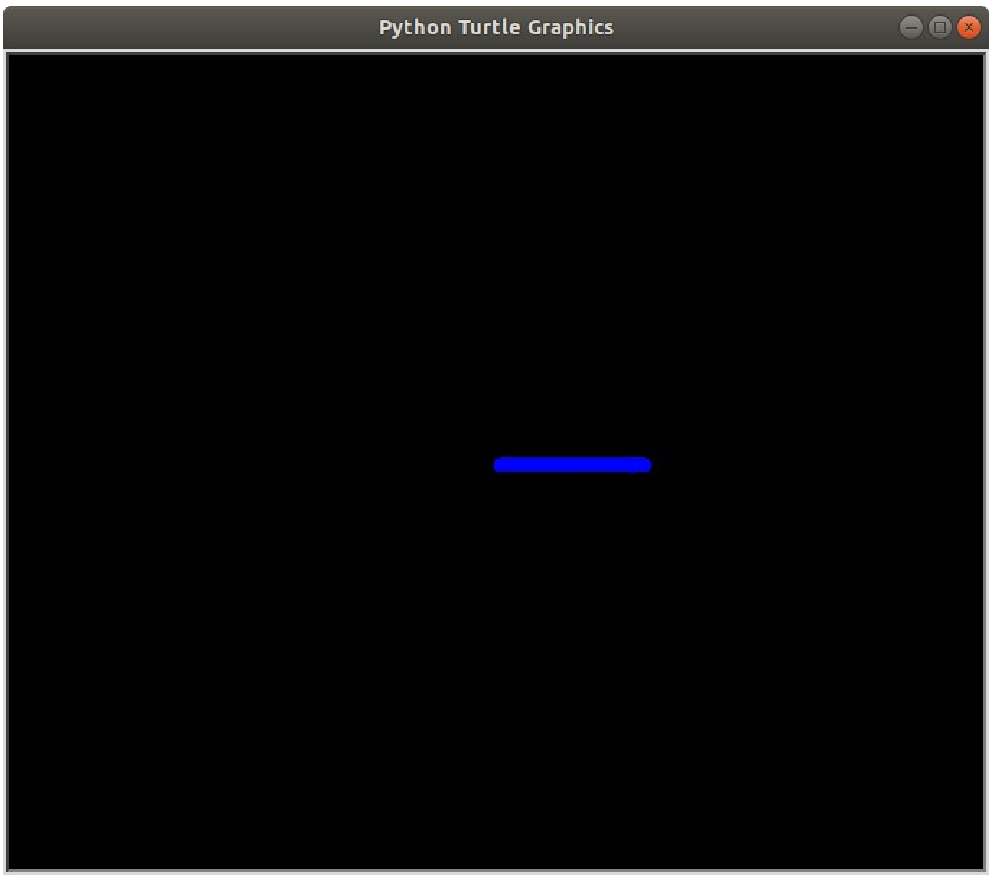
How to Increase the Thickness of a Line

t.pensize() accepts an integer as an argument to set the thickness of the line.

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

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("blue")
t.pensize(10)
t.fd(100)
```

when you run the code snippet above, you will get:



How to Hide the Turtle(Arrow Head)

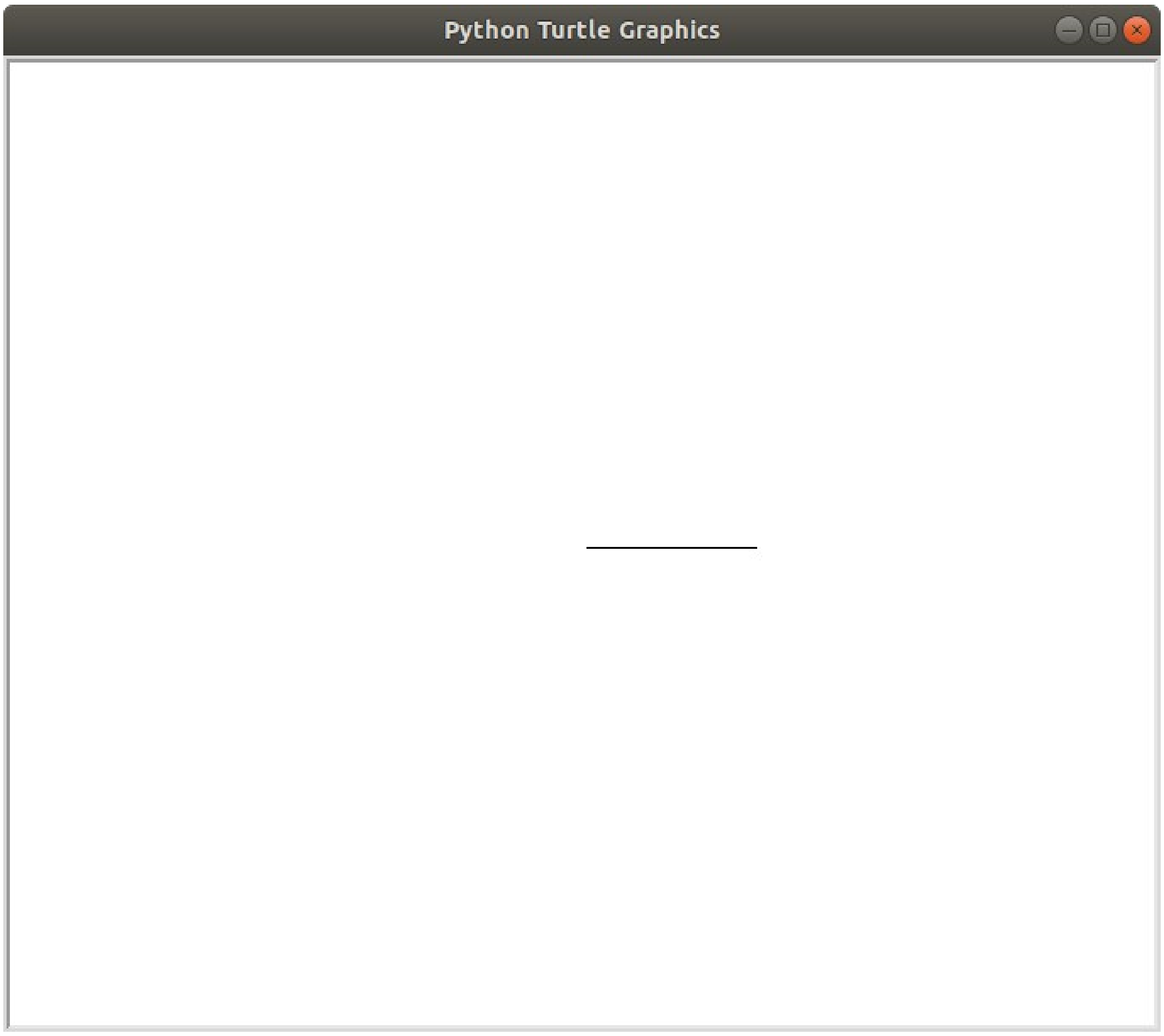
You may not need the turtle to appear in your drawing as you just need a line. To achieve this open a new file editor and save the following code snippet as **draw_line_without_turtle.py**

Hiding turtle

Python

```
from turtle import Turtle
t=Turtle()
t.fd(100)
t.hideturtle()
```

Running this program will give you,



You have learned a lot, so let us do something fun with Python.

Project 1: Draw a square

Write a program to draw a square on the computer screen and save it as **turtle_square.py**

Let the background screen color be black and color of the line drawn red.

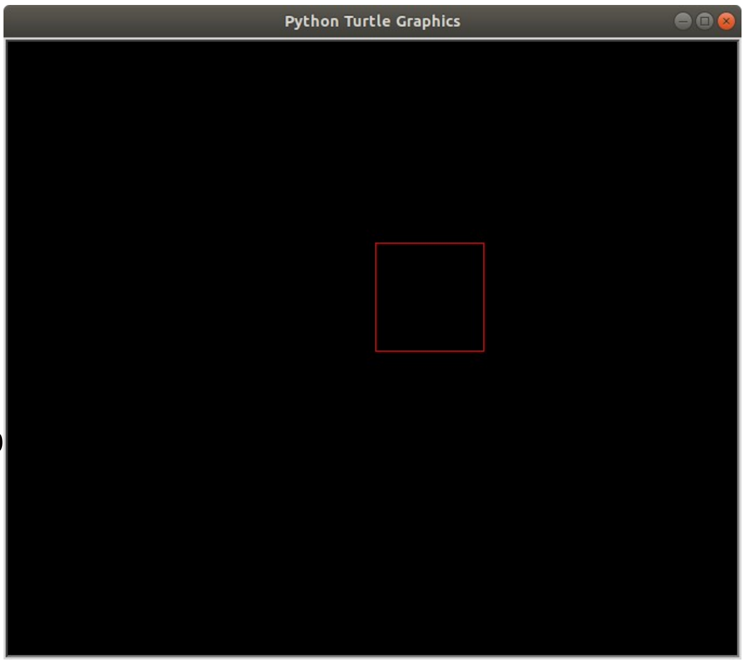
Solution 1:

Drawing a Square

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.hideturtle()
t.fd(100)
t.left(90)
t.fd(100)
t.left(90)
t.fd(100)
t.left(90)
t.fd(100)
```

The above program first changes the window background to black and the turtle to red using ***t.screen.bgcolor("black")*** and ***t.color("red")***.

The turtle is then made invisible using this line of code ***t.hideturtle()***



The turtle moves 100 pixels to the east (***t.fd(100)***) and draws a horizontal line.

It then faces the north by rotating 90degrees(**t.left(90)**) to the left and moves 100 pixels to the north, drawing a vertical line.

t.left(90) makes the turtle face the west by rotating it 90 degrees left. It then moves 100 pixels to the west, drawing a horizontal line.

t.left(90) makes the turtle face the south by rotating it 90 degrees left. It then moves 100 pixels to the south, drawing a vertical line.

The turtle turns 90 degrees left to face the east, it then moves 100 pixels to the east which is the default position of the turtle.

Finally, a square is drawn.

Solution 2: Draw Square

Let’s make our code more readable and maintainable with function and a for loop. Save this as **turtle_square1.py**

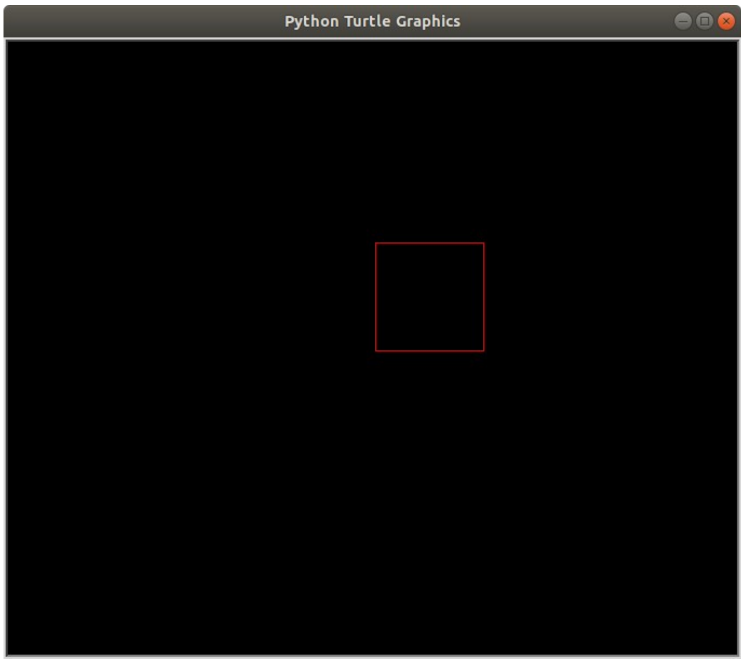
Drawing a square with function and for loop

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.hideturtle()

def square(length):
    for steps in range(4):
      t.fd(length)
      t.left(90)

square(100)
```

Running **turtle_square1.py,** you will get a square on your computer screen.

Basically, the for loop runs through the code below four times.

t.fd(length)
t.left(90)



Project 2: Draw rectangle

Write a new program to draw a rectangle inclined to 30 degrees on the computer screen and save it as **turtle_rectangle.py**

Let the screen background color be black and color of the line drawn red.

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.hideturtle()
t.color("red")

def slanted_rectangle(length,width,angle):
        t.setheading(angle)
        for steps in range(2):
            t.fd(width)
            t.left(90)
            t.fd(length)
            t.left(90)

slanted_rectangle(length=200,angle=45,width=100)
```

Save and run

Write a program to draw a triangle on the computer screen and save it as **turtle_triangle.py**

Let the screen background color be black and color of the line drawn red.

Solution:

Draw a Triangle

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.hideturtle()

def triangle(length,angle=120):
    for steps in range(3):
      t.fd(length)
      t.left(angle)

triangle(200)
```

Save and run

Don’t be over excited yet, there’s a ton of room for improvement in our programs.

How to Draw a Circle

The ***circle()*** method is used to draw a circle and it can accept three arguments which are:

1. radius
2. extent
3. steps

The most important is radius and extent. Though, extent and steps are optional(i.e a circle can still be drawn without including them.)

The radius accepts an integer (whole number) which is the radius of the circle.

The extent accepts an integer (whole number) which is an angle. To cut short the technical explanation, it determines the part of the circle that will be drawn.

If you assign 360, a full circle will be drawn, 180 will draw a semi-circle.

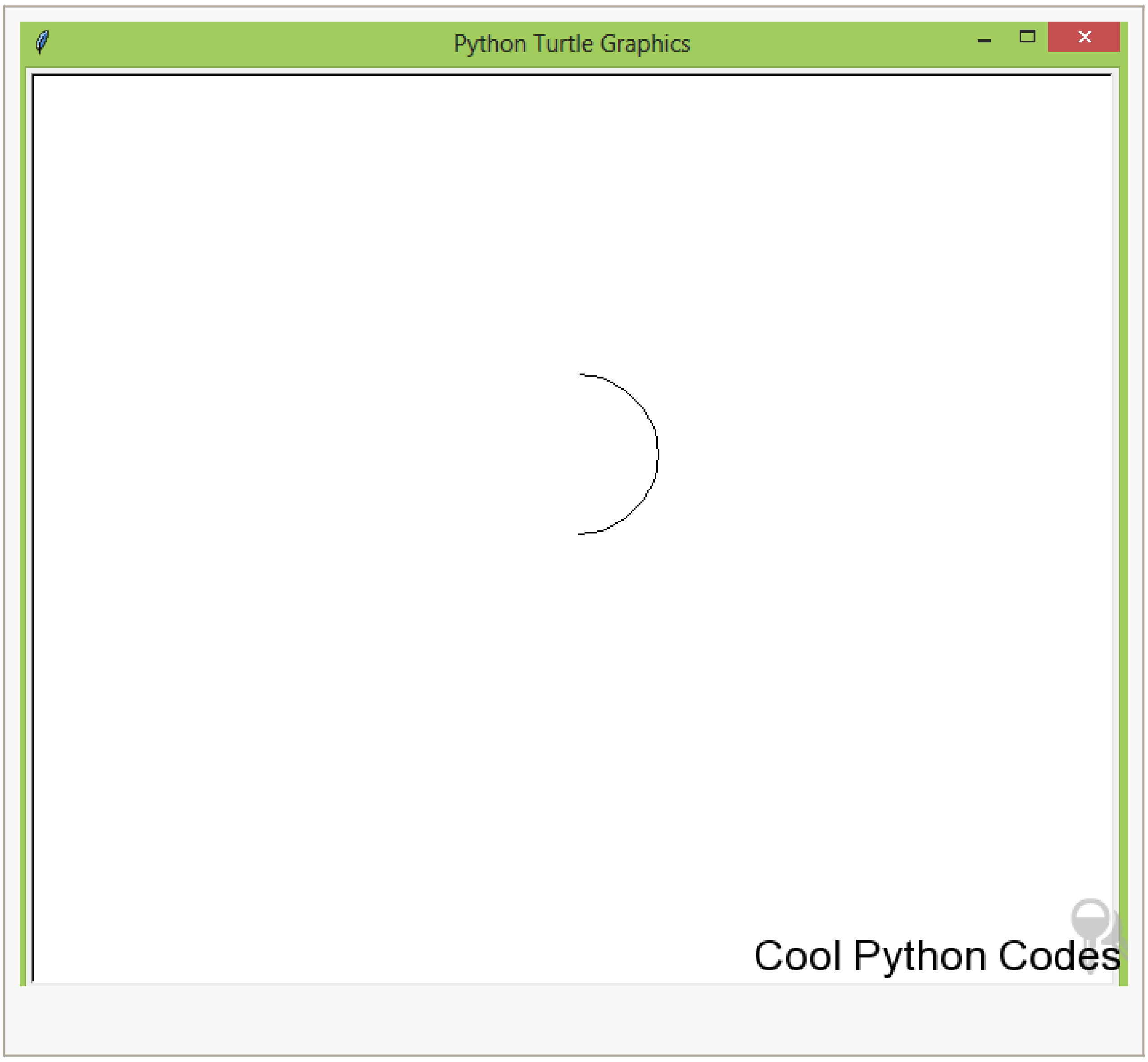
Let’s do some coding. The code snippet below will draw a semi-circle.

Open a new file editor window and type the following code snippet. Save it as **turtle_semi_circle.py**

Semi circle

```
from turtle import Turtle

t=Turtle()

t.circle(50,180)

t.hideturtle()
```

Here’s the output of the code snippet.



Well, let’s do something a bit harder so you can learn more.

```
from turtle import Turtle

t=Turtle()
t.hideturtle()
t.up()
t.setx(50)
t.down()
t.setheading(90)
t.circle(50,180)
```

The only thing strange about this code snippet is ***t.setx().***

The method ***setx()*** positions the turtle on the x-axis leaving the y-axis unchanged.

In addition, the method ***sety()*** positions the turtle on the y-axis leaving the x-axis unchanged. In the next section, we will draw a full circle.

How to Raise Up the Turtle off the Drawing Surface

Imagine the turtle as a paint brush in the hand of an artist. The artist can lift up the brush off the canvas to place it on another position on the canvas.

To do a similar action with the turtle, you use this syntax below,

t.up()

To put back the turtle on the canvas, use this syntax below,

t.down()

Note:

No drawing is done when the turtle is up. Drawing is done when the turtle is down.
Let me explain the following syntax by writing cool programs

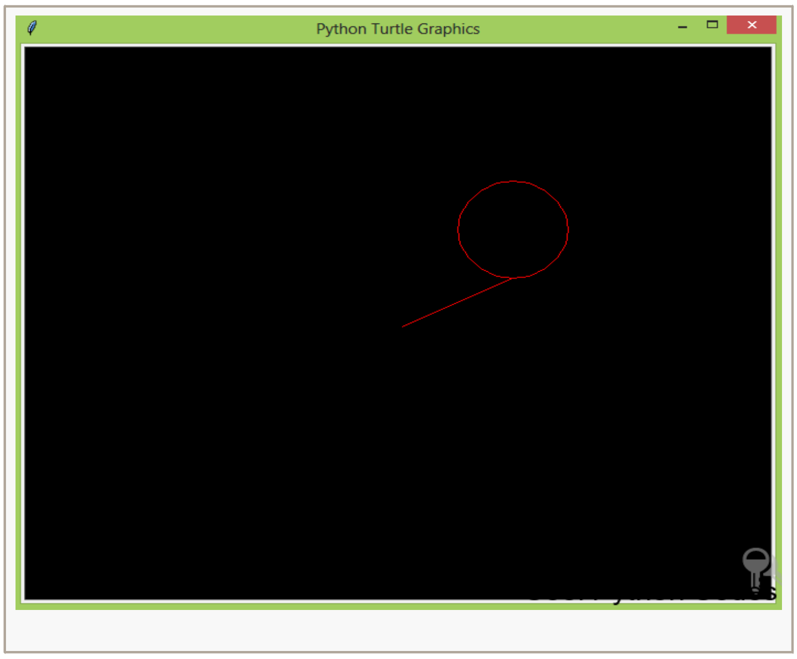
up()
down()
goto(x,y)
circle(radius)

Code snippet 1:

Open a new file editor window and type the following code snippet. Save it as **code_snippet_1.py**

Python

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.goto(100,50)
t.circle(50)
t.hideturtle()
```



If you run the code above you will get;

From the diagram above you can observe that the turtle moved to an absolute position

t.goto(100,50))

and a circle with a radius of 50 is drawn
t.circle(50))

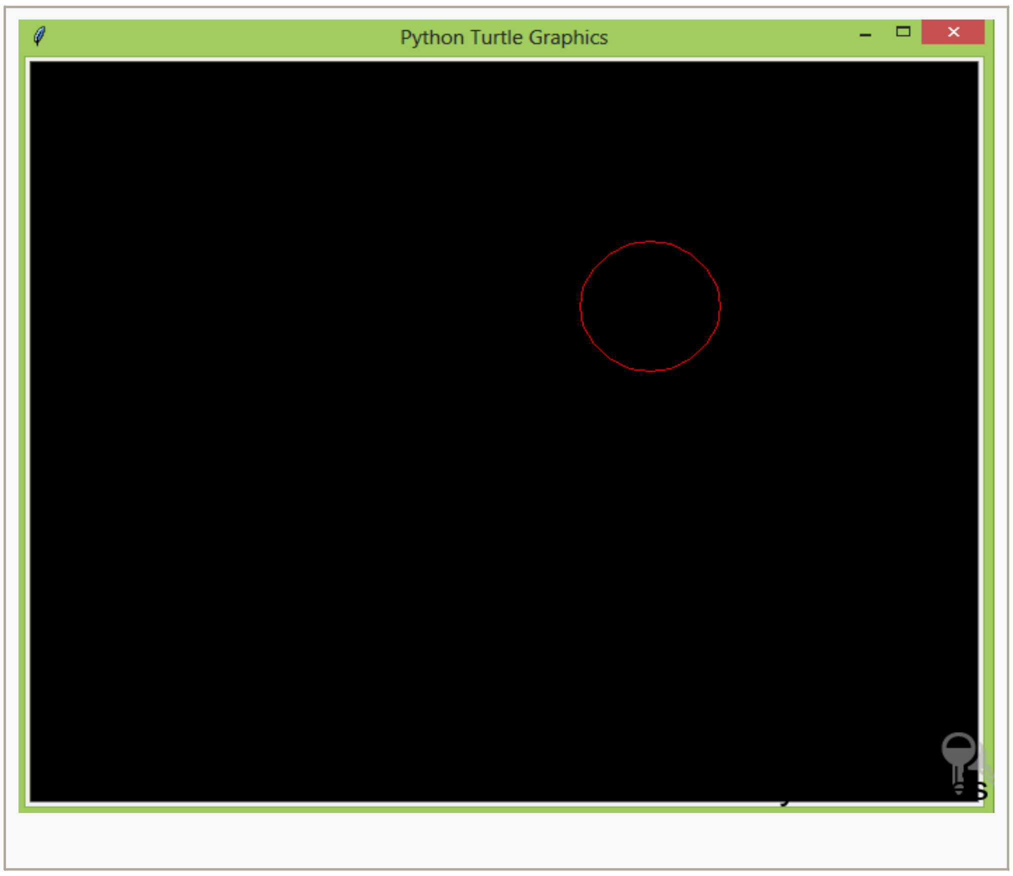
The next code snippet will remove the line in the diagram above.

Code snippet 2:

Open a new file editor window and type the following code snippet. Save it as **code_snippet_2.py**

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.up()
t.goto(100,50)
t.down()
t.circle(50)
t.hideturtle()
```

If you run the code above you will get;



t.up() – raises the turtle up and no drawing is done when the turtle is up.
t.goto(100,50) – moves the turtle to an absolute position.
t.down() – brings down the turtle.
t.circle(50) – draws a circle with a radius of the value 50

you can replace t.up(), t.goto(100,50), t.down() with the method ***setposition()***

The setposition() method accepts two arguments which are the coordinates x and y.

This method moves the turtle to an absolute position. If the turtle pen is down, a line will be drawn to the specified position.

How to Add colors in Your Drawing

We have been drawing many shapes but we have not applied colors to our drawing.

Here are some ways to apply colors to your drawing.

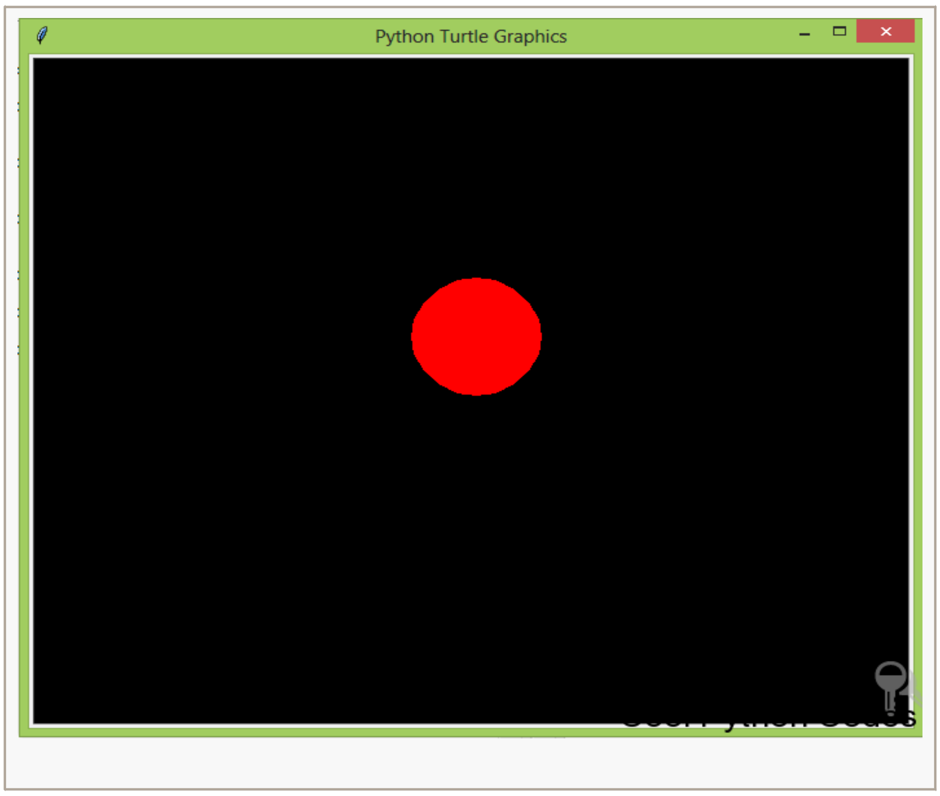
t.begin_fill() and t.end_fill() –

t.begin_fill() is called before drawing a shape to be filled with the color of the turtle. *t.end_fill()* fills the shape drawn with the color of the turtle after *t.begin_fill()* has been called. Open a new file editor window and type the following code snippet. Save it as **red_circle.py** red circle

Python

```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("red")
t.begin_fill()
t.circle(50)
t.end_fill()
t.hideturtle()
```

If you run the code above you will get:



t.fillcolor()

This meth0d changes the color of the turtle only, when the turtle moves, the color path remains the default color (black) .

The color can be passed as a string such as:

t.fillcolor(“blue”)

The next thing we are going to do will blow your mind.

Random Walk with Turtle

Open a new file editor window and type the following code snippet. Save it as **turtle_random_walk.py**

turtle drawing randomly

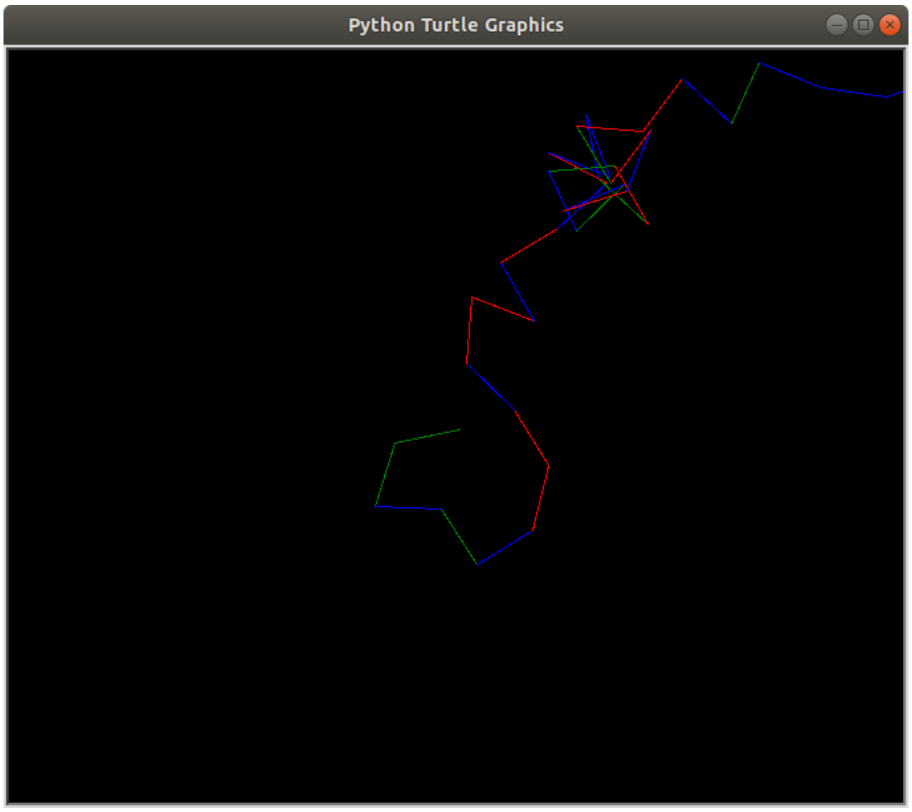
```
from turtle import Turtle
import random

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

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(100,50)
```

My output of the program is something like this:

Note:

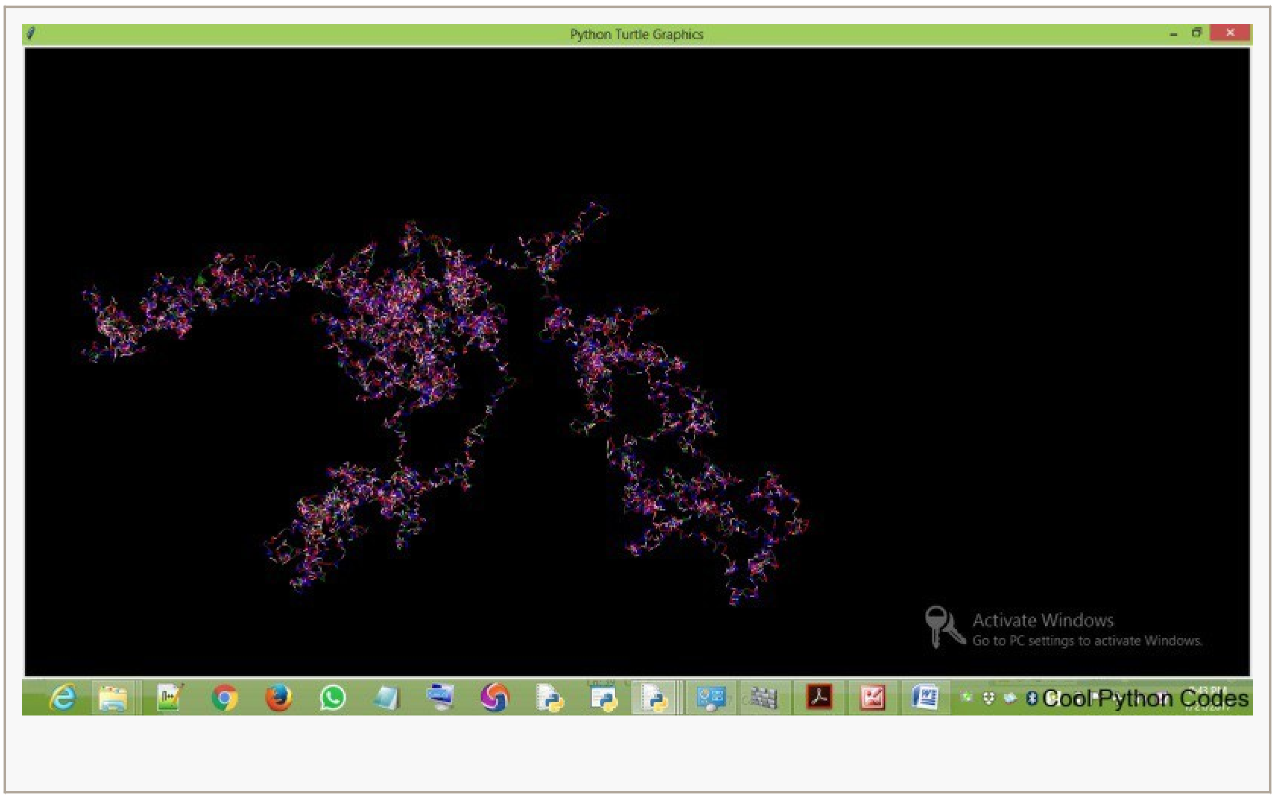
We can’t get the same result. Why?
This is because I use the **random** module.

A **random** module picks random element in a sequence.



How to Increase the Speed of the Turtle

Imagine how long it will take for your turtle to draw something like this:



It would take you a lot of time. But here’s the best part of turtle,
You can increase the speed of the turtle. How then?

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

So to increase the speed of the program (**turtle_random_walk.py**) include ***t.speed(0)*** before defining the function randomwalk.

How to draw a dot

The ***dot()*** method receives two optional arguments, which are:

- The size of a dot
- The color of the dot

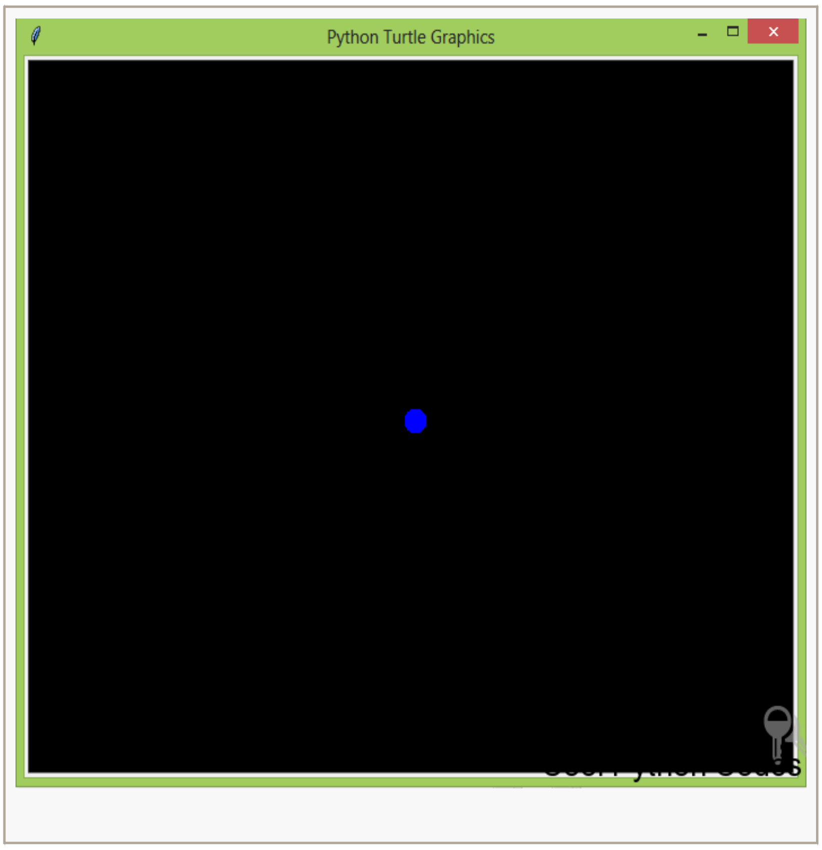
Here is a simple dot of size 20 and blue in color.

Here is the code snippet.

Python turtle code snippet to draw a dot

```
from turtle import Turtle
t = Turtle()
t.screen.bgcolor("black")
t.hideturtle()
t.dot(20,"blue")
```

So we have been drawing shapes on our computer screen, and am getting really bored of it. Let’s do something a bit different by writing text on the computer screen.



How to Write Text with Python Turtle

To write on your computer screen, use the ***write()*** method. The write() method accepts four arguments, which are:

arg– this is the string object of the text you want to write to the TurtleScreen.

move– this argument is optional and it can only be assigned to the value “True” or “False”. When True a line will be drawn below the text.

align– you assign this argument either to the left, right or center. This argument is also optional

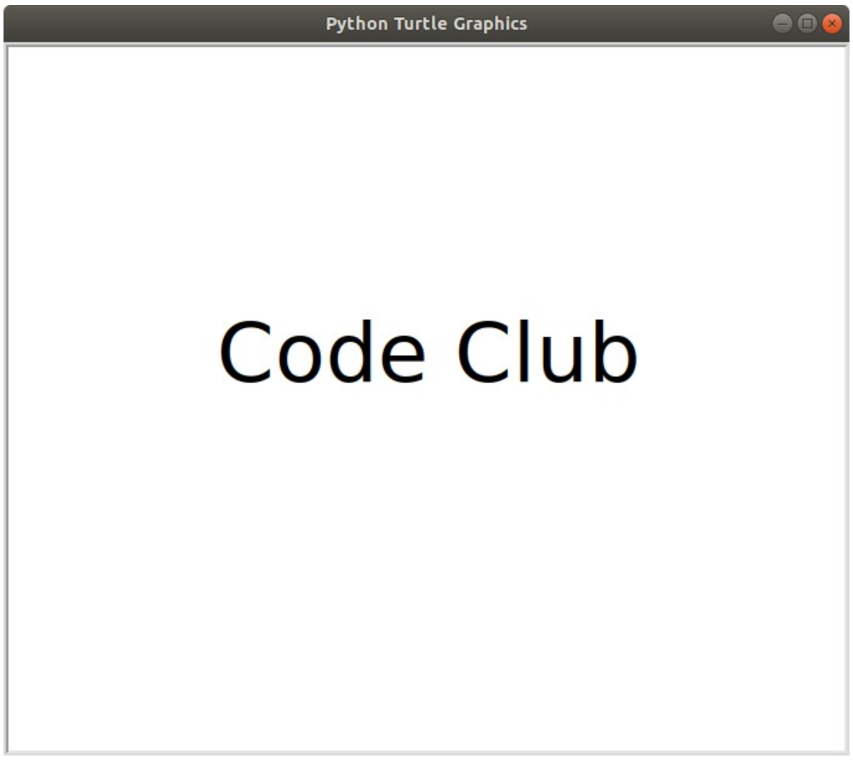
font– this argument basically determines the font name, font size, font type of your text.
To understand this better, let’s write some codes.

Write a program to display Cool Python Codes on the turtle screen.

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

```
from turtle import Turtle
t = Turtle()
t.hideturtle()
t.write("Cool Python Codes",move=True,align="center",font=("Freestyle  Script",50,"normal"))
```

Here is the output of the program.



Lets put it all together..

```
from turtle import Turtle
t = Turtle()
t.screen.bgcolor("black")
t.color("white")

def square(length):
        for steps in range(4):
                t.fd(length)
                t.left(90)

def draw_square(x, y, length):
        t.hideturtle()
        t.up()
        t.goto(x, y)
        t.down()
        t.begin_fill()
        square(length)
        t.end_fill()

def rectangle(length, width):
        for steps in range(2):
        t.fd(width)
        t.left(90)
        t.fd(length)
        t.left(90)

def draw_rectangle(length, width, x, y):
        t.hideturtle()
        t.up()
        t.goto(x, y)
        t.down()
        t.begin_fill()
        rectangle(length, width)
        t.end_fill()

t.write("Longlevens Code Club", move=True, align='center',font=('Cambria', 18, 'normal'))

draw_square(-135, -20, 20)
draw_square(-135, 30, 20)
draw_square(-170, 0, 30)

t.screen.exitonclick()

draw_rectangle(70,10,-170,-0)
draw_rectangle(10,70,-170,-30)
draw_rectangle(10,70,-170,50)
```

onclick() method

This method calls a function whenever the mouse is used to click on a particular coordinate. It accepts three arguments which are:

- ***fun***– this is a function with two arguments, to which will be assigned the coordinates(x,y) of the clicked point on the canvas.
- ***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

binding. I usually use one argument which is the function I want to call when I click my mouse.
Here is a program that draws a circle when you click on the turtle. Save the code snippet as
click_to_draw_circle.py

Draw a circle with a mouse click

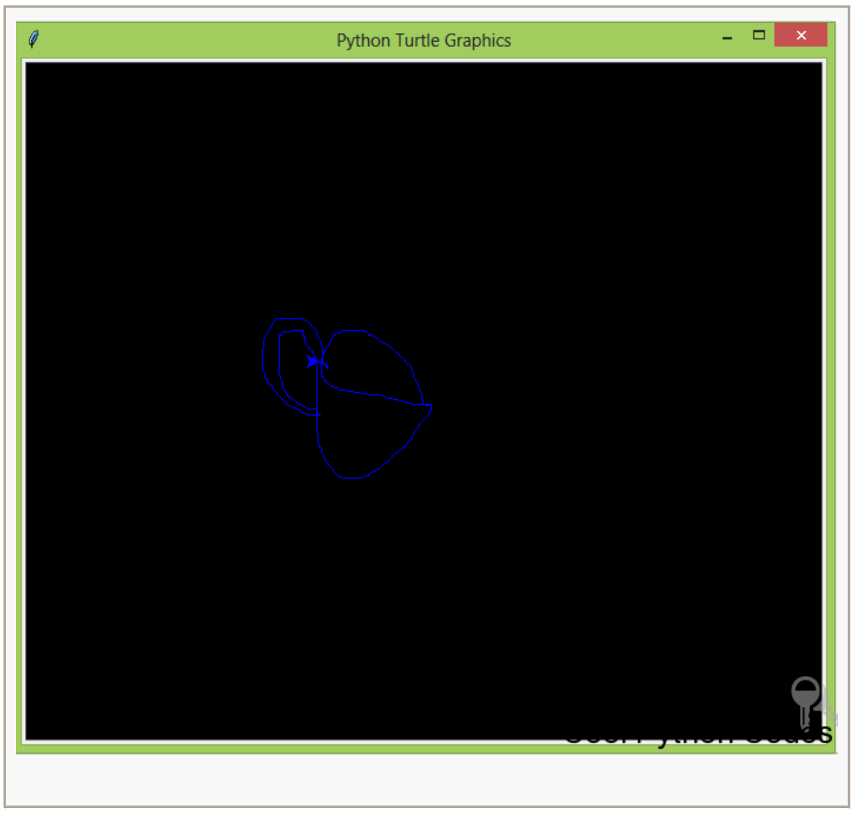
```
from turtle import Turtle
t=Turtle()
t.screen.bgcolor("black")
t.color("orange")

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

t.onclick(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.
The code snippet below is saved as
turtle_ondrag.py. Python

```
from turtle import Turtle

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

def goto(x,y):
    t.goto(x,y)

t.ondrag(goto)
```

Here's what I drew using the program above.



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.

Here is the code snippet and it is saved as **turtle_onrelease.py**

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

def draw_circle(x,y):
   t.circle(50)

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

t.onclick(draw_circle)
t.onrelease(erase_drawing)
```

Note:

You have to leave your hand on the left-hand button of the mouse until the circle is drawn on the screen.

When the circle has been drawn, you can then remove your hand from your mouse. Once you remove your hand from the mouse, the circle disappears.

This circle disappears because of the ***clear()*** method. The clear() method deletes the turtle's drawing on the screen.

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

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**

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

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()

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()

keyboard_commands()
t.screen.mainloop()
```

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. One more last thing before I end this tutorial.

How to Change the Title of the Turtle Screen

The default title of the turtle screen is “Python Turtle Graphics”, which can be boring.

It will be fun if your name or your friend’s name is on it. Your friends will know that you’re a Pro programmer. To change the title of the turtle screen, use the method ***screen.title().*** This method only accepts a string.
To illustrate this method, open a new file editor and save the following code snippet as **turtle_screen_title.py**

Python

```
from turtle import Turtle

t=Turtle()

t.screen.title("Code Club”)
```

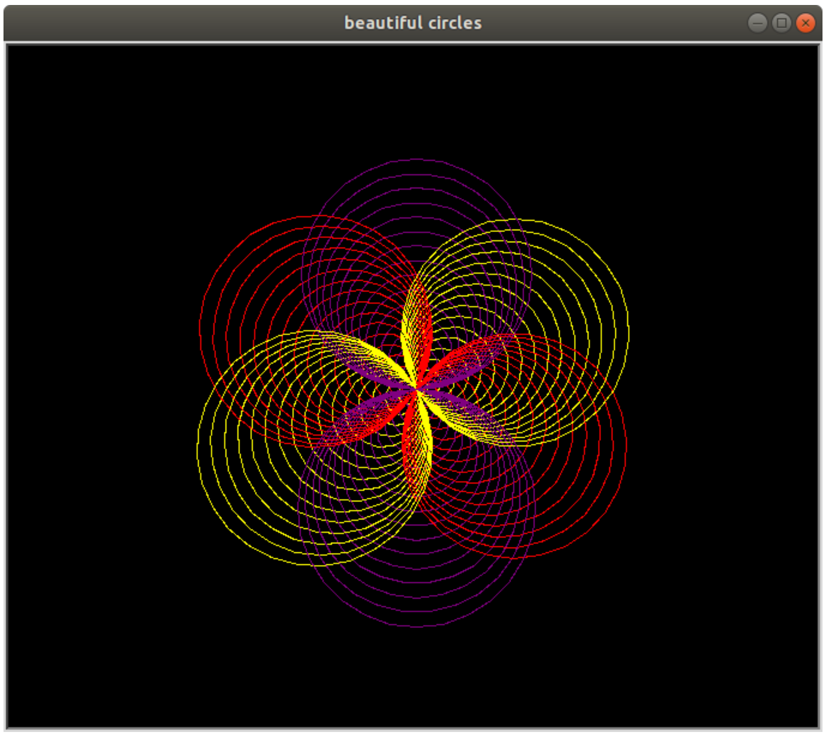
Save and Run

Try and study the code snippet and do a lot of experiments to come up with new designs.

The below code snippet is saved as **beautiful_circles.py**

```
from turtle import Turtle

t=Turtle()
t.screen.bgcolor("black")
t.screen.title(“beautiful circles”)
colors=["red","yellow","purple"]
t.screen.tracer(0,0)

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

t.screen.exitonclick()
t.screen.mainloop()
```

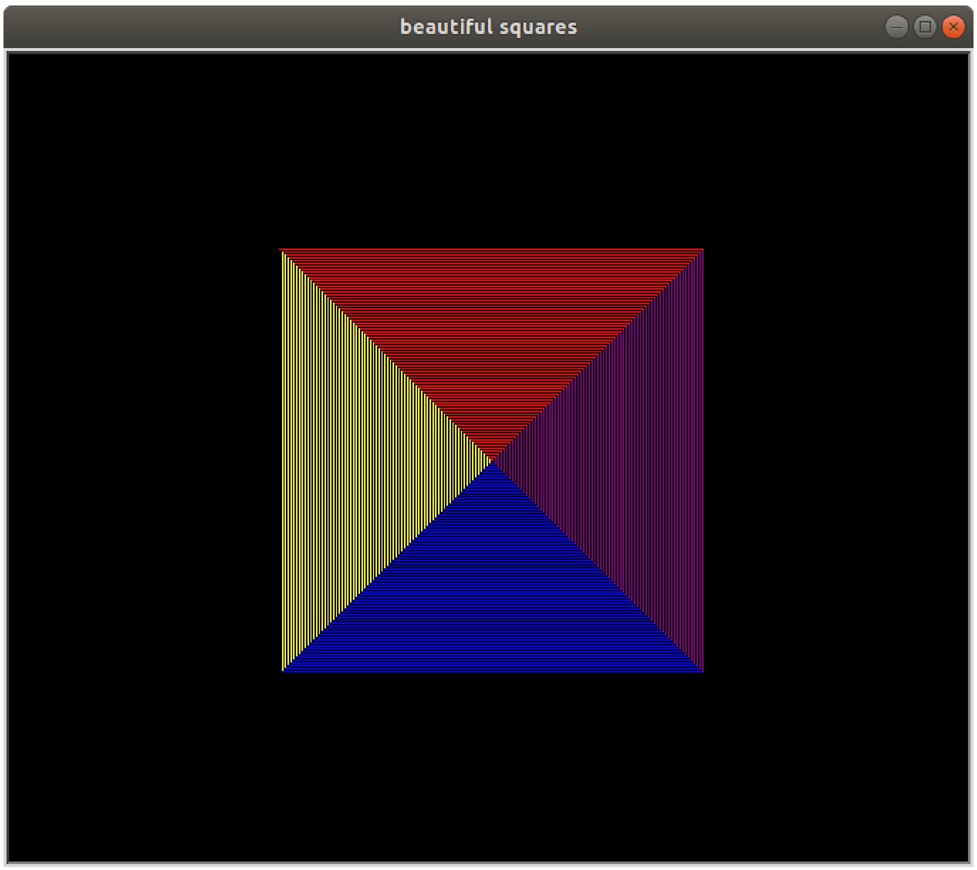
Here’s the output of the above code snippet.



The below code snippet is saved as **beautiful_square.py**

```
from turtle import Turtle

t=Turtle()
t.screen.bgcolor("black")
t.screen.title("beautiful squares")

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

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

t.screen.exitonclick()
t.screen.mainloop()
```

Here’s the output of the code snippet above.

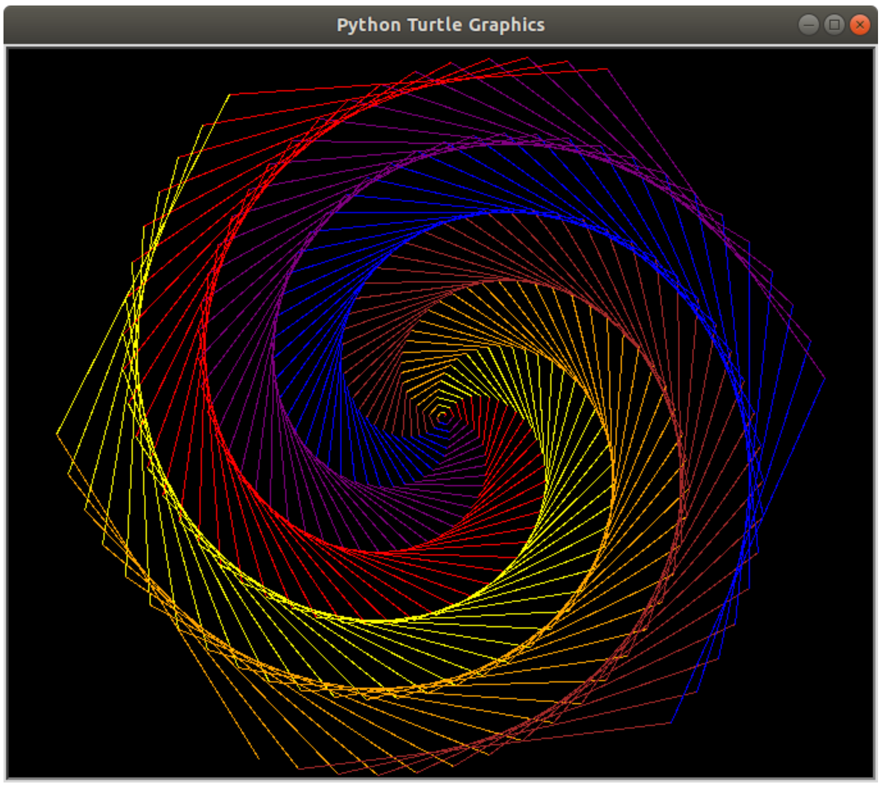


The below code snippet is saved as **beautiful_spiral.py**

```
from turtle import Turtle

t=Turtle()
```

```
t.screen.bgcolor("black")
colors=["blue","purple","red","yellow","orange","brown"]
t.screen.tracer(0,0)

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

t.screen.exitonclick()
 t.screen.mainloop()
```

Here’s the output of the above code snippet.


Python Turtle Graphics