**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.

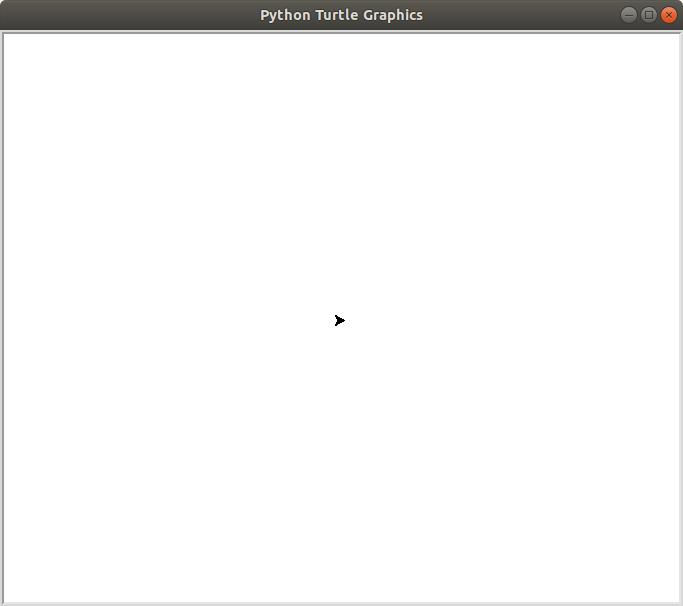
You can see an arrow head at the middle point of the window pointing to the east which is the default position. This location is called home.

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()*

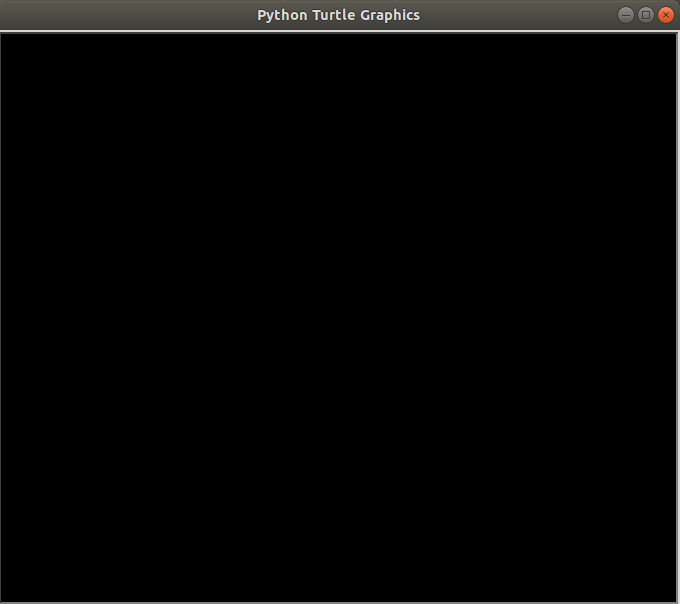
*t.screen.bgcolor(‘black’)*

*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*

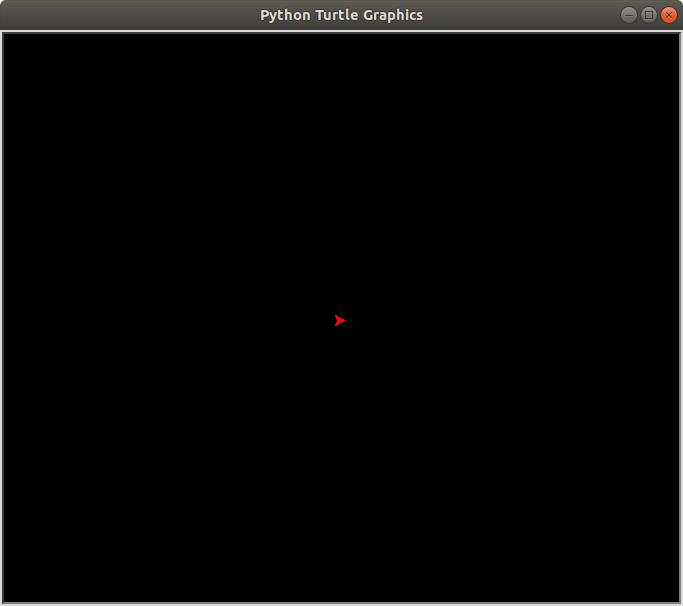
*t=Turtle()*

*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*

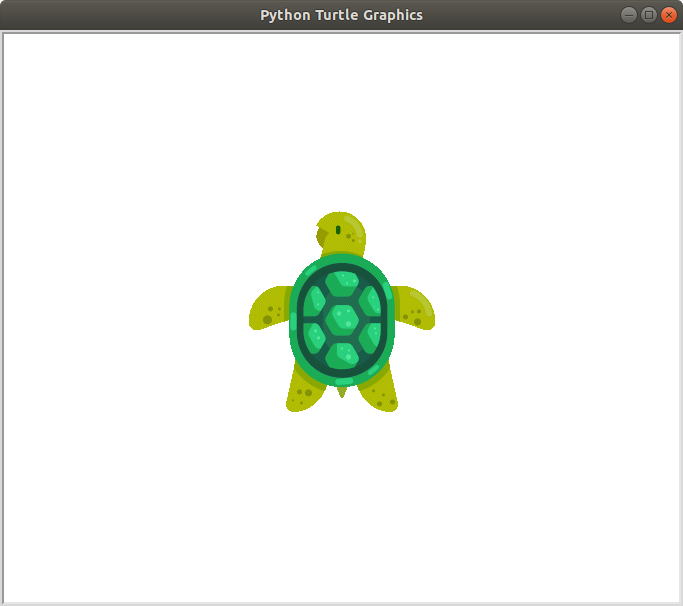
*t=Turtle()*

*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)*

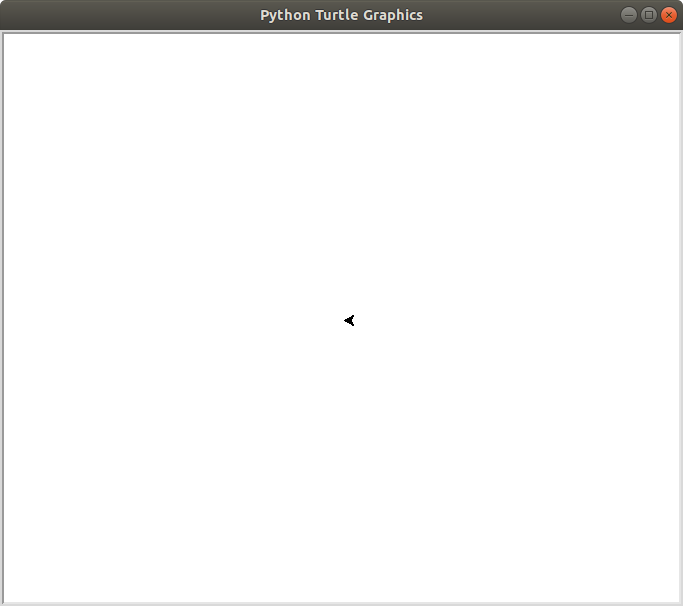
*t.screen.exitonclick()*

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*

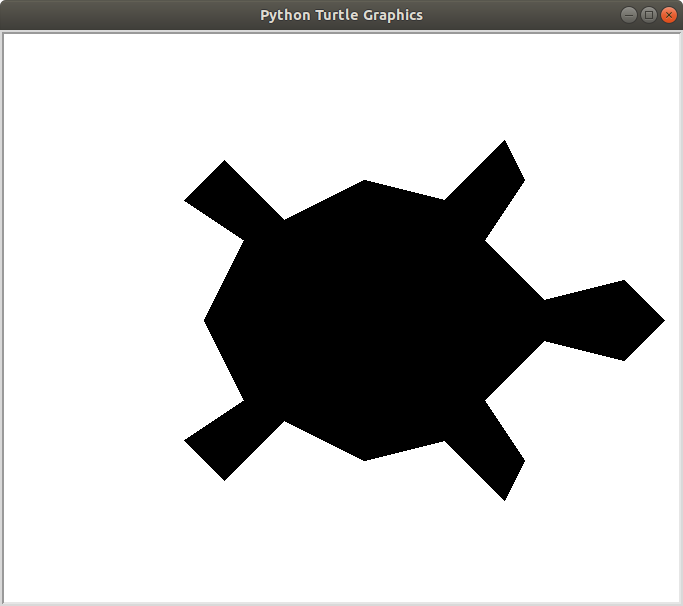
*t=Turtle()*

*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

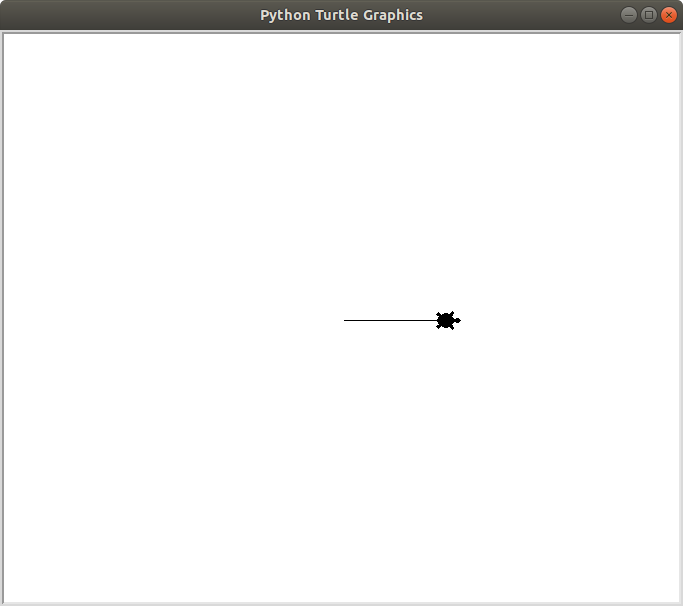
***t.goto(x,y)***

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()*

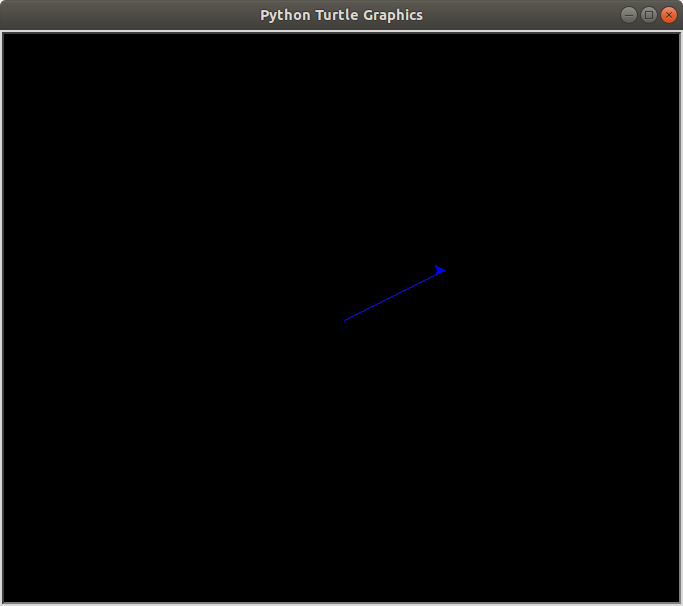
When you run the program above you will get:

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*

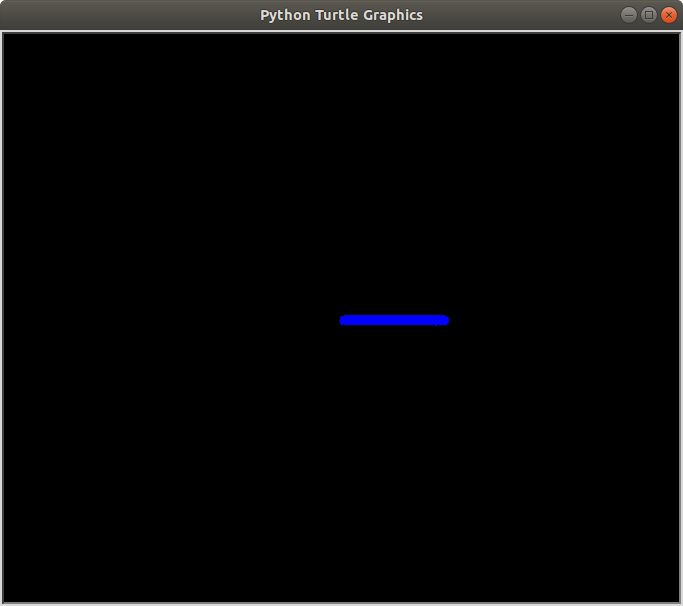
*t=Turtle()*

*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

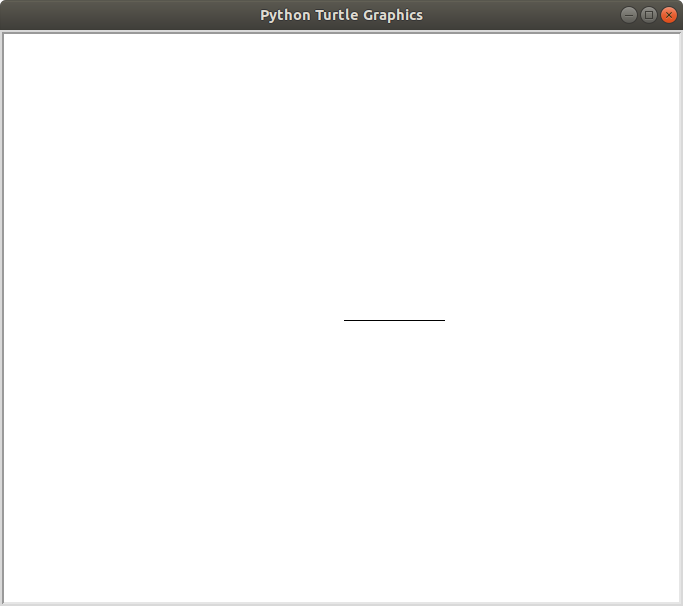
from turtle import Turtle

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:**

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: for a beginner that doesn’t know** [**Python function**](http://coolpythoncodes.com/python-function/) **and for loop**

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

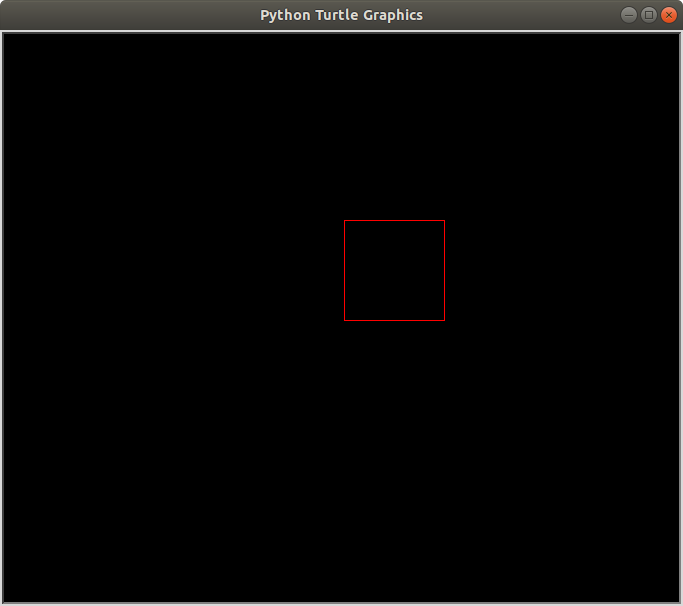
***t.screen.bgcolor(“black”)*** and ***t.color(“red”)****.*

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:**

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):

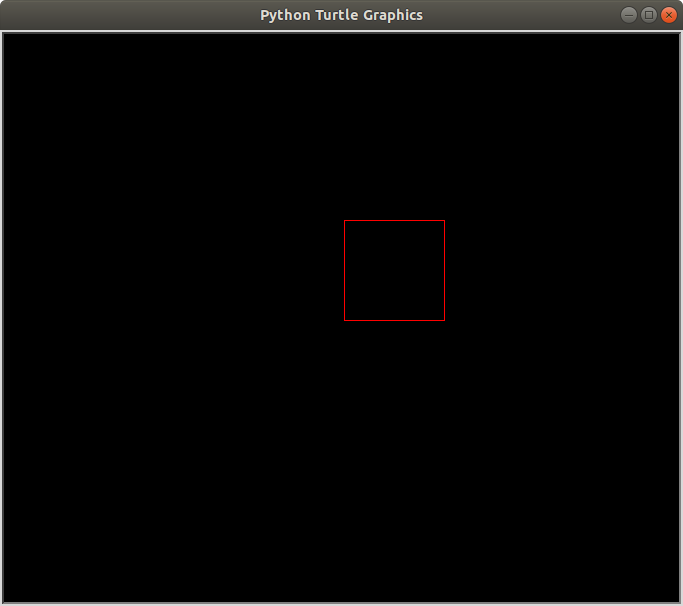
for steps in range(4):

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:**

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.

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

4 t.hideturtle()

5 t.color("red")

6

7 def slanted\_rectangle(length,width,angle):

8 t.setheading(angle)

9 for steps in range(2):

10 t.fd(width)

11 t.left(90)

12 t.fd(length)

13 t.left(90)

14

15 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

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

4 t.color("red")

5 t.hideturtle()

6

7 def triangle(length,angle=120):

8 for steps in range(3):

9 t.fd(length)

10 t.left(angle)

11

12 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 which is the radius of the circle.

The extent accepts an integer 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. (Do you now understand why I said extent represents an angle?)

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

1 from turtle import Turtle

2

3 t=Turtle()

4

5 t.circle(50,180)

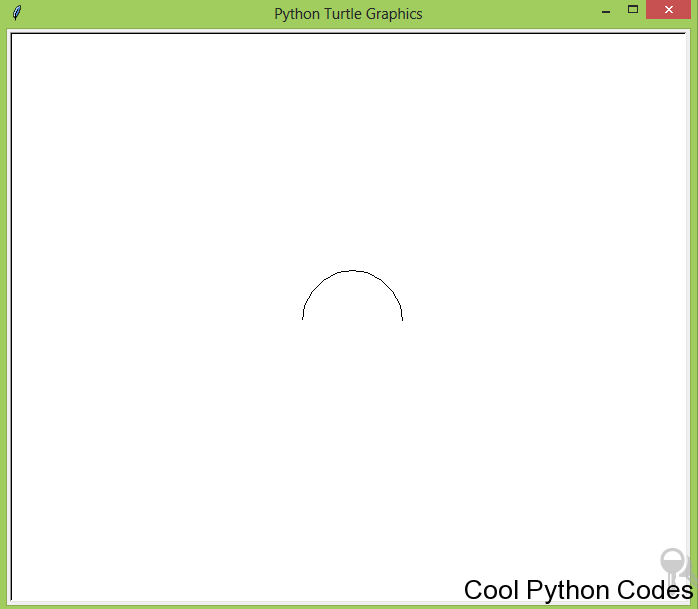
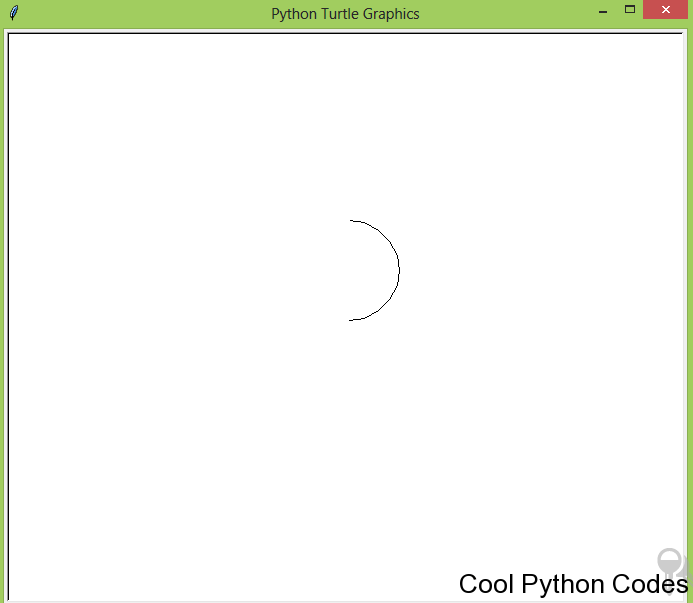
6

7 t.hideturtle()

Here’s the output of the code snippet.

Semi circle

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



Semi Circle

If you could not do it, don’t worry.

Open a new file editor window and type the following code snippet. Save it as **semi\_circle1.py**

Python

1 from turtle import Turtle

2

3 t=Turtle()

4

5 t.hideturtle()

6

7 t.up()

8

9 t.setx(50)

10

11 t.down()

12

13 t.setheading(90)

14

15 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)*

I want you to observe the following code snippet and the diagram.

**Code snippet 1:**

Open a new file editor window and type the following code snippet. Save it as **code\_snippet\_1.py**

Python

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

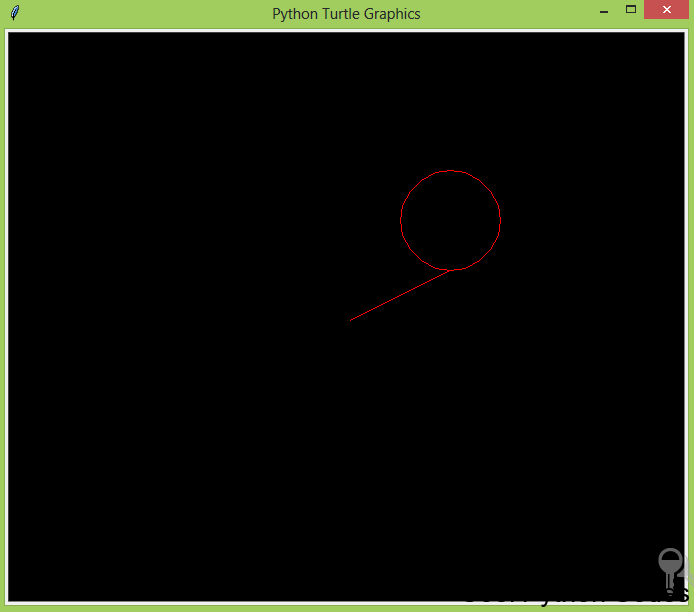
4 t.color("red")

5 t.goto(100,50)

6 t.circle(50)

7 t.hideturtle()

If you run the code above you will get;



code snippet 1 screenshot

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**

Python

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

4 t.color("red")

5 t.up()

6 t.goto(100,50)

7 t.down()

8 t.circle(50)

9 t.hideturtle()

If you run the code above you will get;

code snippet 2 screenshot

***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(), t.circle(50) 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

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

4 t.color("red")

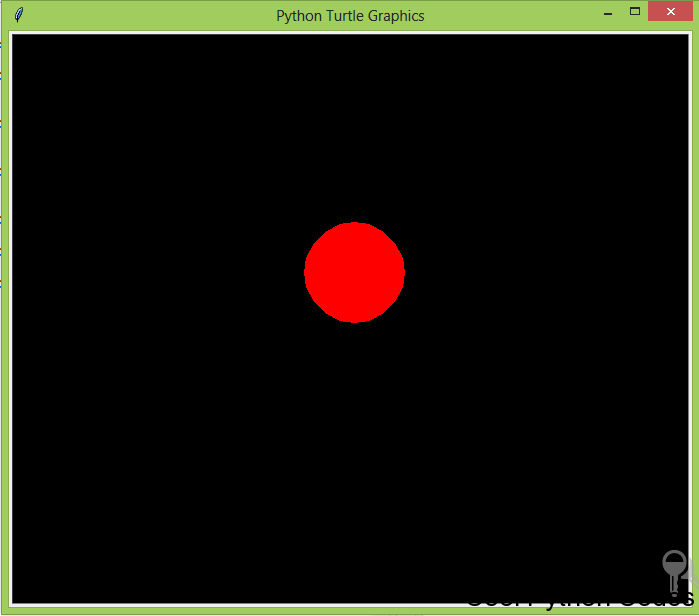
5 t.begin\_fill()

6 t.circle(50)

7 t.end\_fill()

8 t.hideturtle()

If you run the code above you will get:



Red circle

**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))

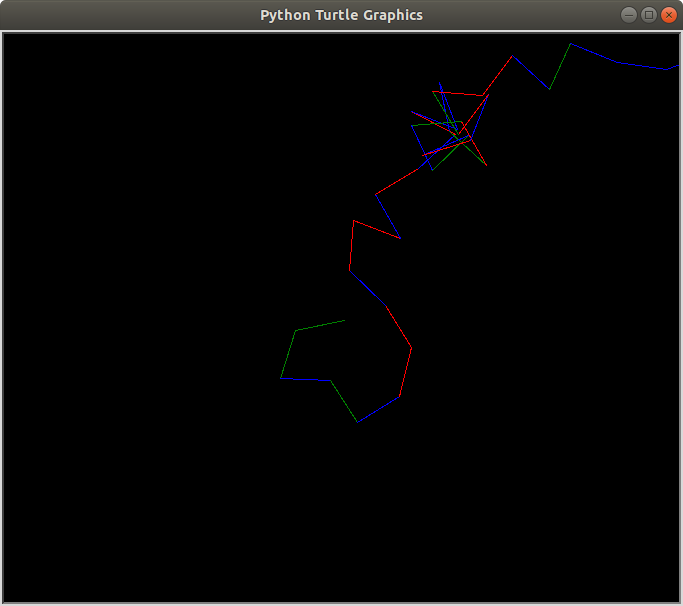
t.color(random.choice(["blue","red","green"]))

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.

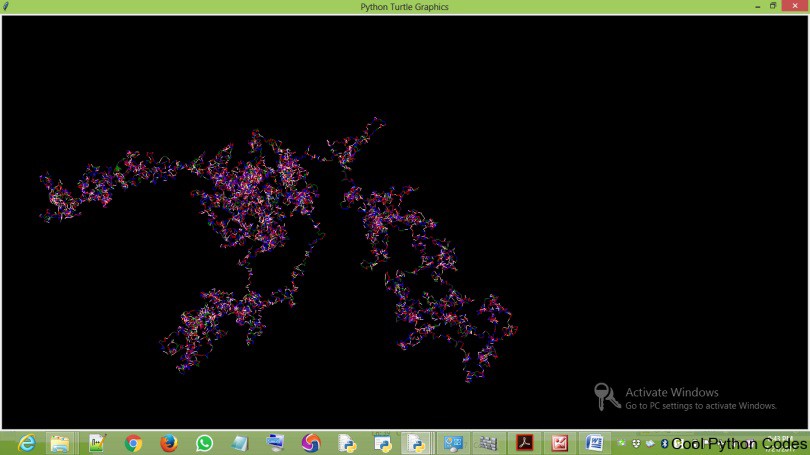
A **random** module picks random element in a sequence.

**How to Increase the Speed of the Turtle**

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

Random drawing

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.

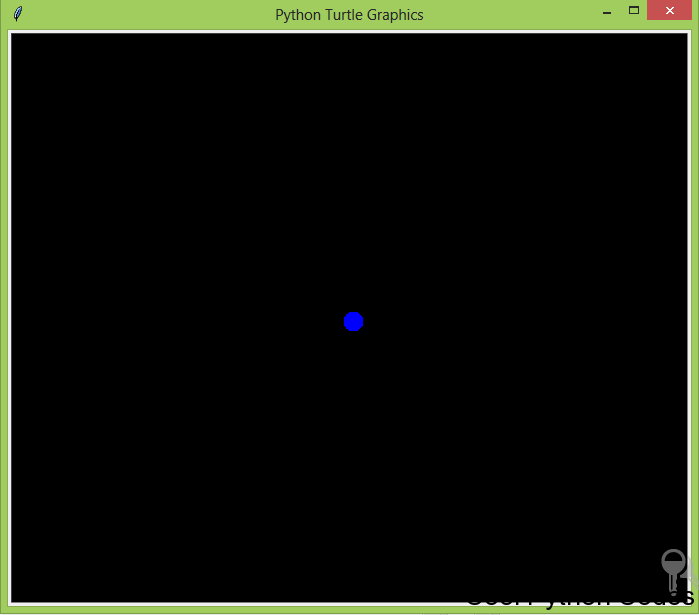
**How to draw a dot**

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.



A dot

Here is the code snippet.

Python turtle code snippet to draw a dot

Python

1 from turtle import Turtle

2 t = Turtle()

3 t.screen.bgcolor("black")

4 t.hideturtle()

5 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**

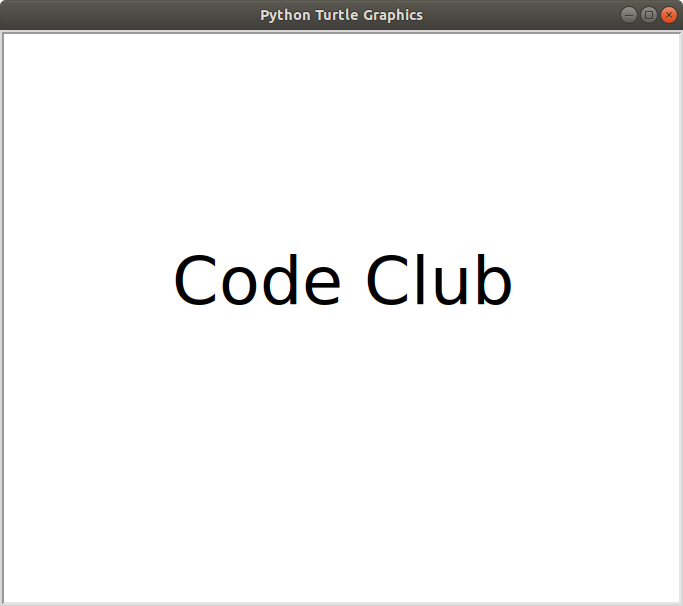
from turtle import Turtle

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

Python

1 from turtle import Turtle

2 t=Turtle()

3 t.screen.bgcolor("black")

4 t.color("orange")

5

6 def circle(x,y):

7 t.circle(60)

8

9 t.onclick(circle)

**Note:**

The function name “circle” has two arguments, x and y which are the coordinates you will have to click 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. The code snippet below is saved as **turtle\_ondrag.py.** Python

1 from turtle import Turtle

2

3 t=Turtle()

4 t.screen.bgcolor("black")

5 t.color("blue")

6

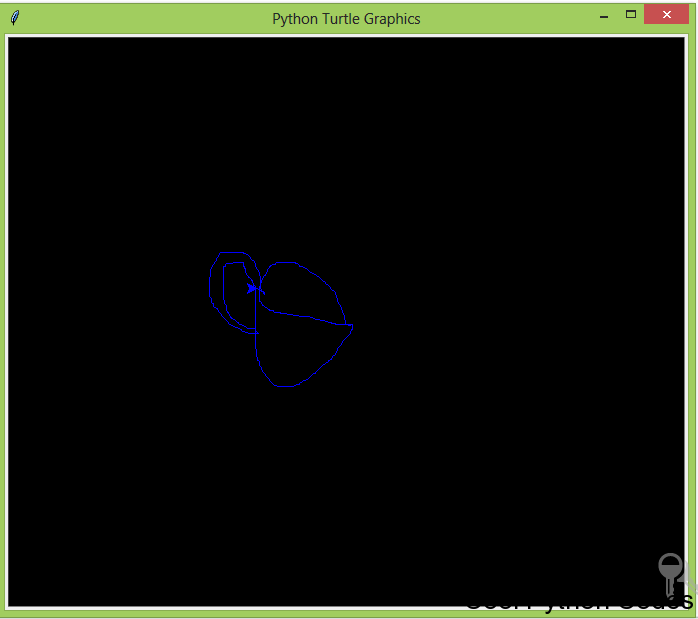
7 def goto(x,y):

8 t.goto(x,y)

9

10 t.ondrag(goto)

Here’s what I drew using the program above.



I drew a cup.

**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**

1 from turtle import Turtle

2

3 t=Turtle()

4

5 def draw\_circle(x,y):

6 t.circle(50)

7

8 def erase\_drawing(x,y):

9 t.clear()

10

11 t.onclick(draw\_circle)

12 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

**listen()**

I will avoid using any technical word because I’m as well bored with them. Basically, without including this method, the turtle won’t obey your commands.

The listen() method is like a cane you use on a child to obey your instruction. Alright, let’s write some codes.

**Project 6:**

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

Python

1 from turtle import Turtle

2 t=Turtle()

3

4 def up():

5 if not(t.heading() == 90):

6 t.setheading(90)

7 t.fd(50)

8 else:

9 t.fd(50)

10

11 def down():

12 if not(t.heading() == 270):

13 t.setheading(270)

14 t.fd(50)

15 else:

16 t.fd(50)

17

18 def right():

19 if not (t.heading() == 0):

20 t.setheading(0)

21 t.fd(50)

22 else:

23 t.fd(50)

24

25 def left():

26 if not (t.heading() ==180):

27 t.setheading(180)

28 t.fd(50)

29 else:

30 t.fd(50)

31

32 def undo\_button():

33 t.undo()

34

35 def keyboard\_commands():

36 t.screen.onkey(up,"Up")

37 t.screen.onkey(down,"Down")

38 t.screen.onkey(right,"Right")

39 t.screen.onkey(left,"Left")

40 t.screen.onkey(undo\_button,"End")

41 t.screen.listen()

42

43 keyboard\_commands()

44 t.screen.mainloop()

Don’t let the long code snippet to confuse you. I will quicky explain the program above.

I mentioned early that :

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

Alright!!!…

Now the “if”, “ else “ and “not” tells the turtle that if it’s not facing the desired direction, that it should change the direction.

**t.heading()** returns the current turtle’s current heading.

**t.undo()** undo the last action of the turtle.

The key commands are “Up”, “Down”, “Left”, “Right” and “End” which are your direction keys and end key on the keyboard. Note that they all start with capital letters.

The rest of the code snippets are easy to understand.

**stamp() method**

This method basically pastes the turtle shape on the canvas on the current turtle’s position.

To illustrate this, open a new file editor and save the following code snippet as **turtle\_stamp.py**

stamp method() Python

1 from turtle import Turtle

2

3 t=Turtle()

4 t.up()

5 t.setx(-50)

6 t.down()

7 t.hideturtle()

8

9 for i in range(4):

10 t.fd(50)

11 t.stamp()

Here’s the output of the above code snippet.

Observing the above screenshot, you will see that the turtle shape appeared 4 times. This makes the concept of for loop more practical. For every 50pixels to the east, a turtle shape will appear.

**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

1 from turtle import Turtle

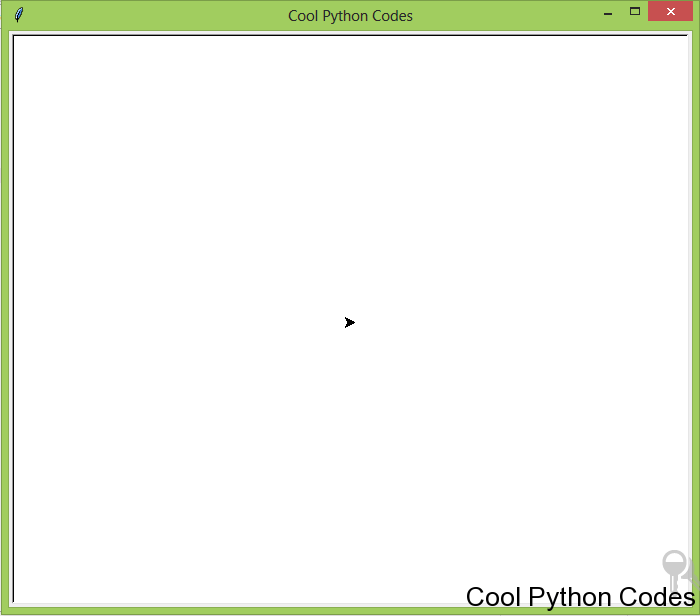
2

3 t=Turtle()

4

5 t.screen.title("Cool Python codes")

Here is the output of the program above.



How to change the title of Python turtle.

Here are some amazing drawings I did with Python Turtle including the code snippet.

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**

Circle spiral

Python

1 from turtle import Turtle

2

3 t=Turtle()

4 t.screen.bgcolor("black")

5 colors=["red","yellow","purple"]

6 t.screen.tracer(0,0)

7

8 for x in range(100):

9 t.circle(x)

10 t.color(colors[x%3])

11 t.left(60)

12

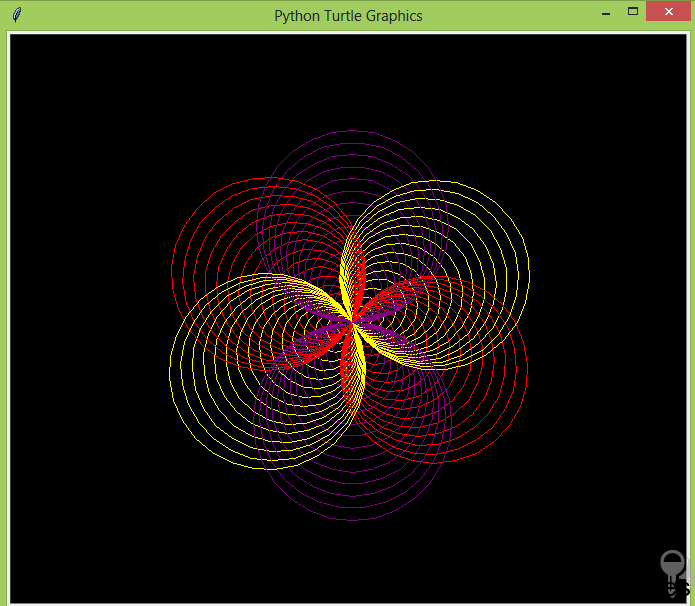
13 t.screen.exitonclick()

14 t.screen.mainloop()

Here’s the output of the above code snippet.

Beautiful circles with Python Turtle

The below code snippet is saved as **beautiful\_square.py**



Beautiful Square

Python

1 from turtle import Turtle

2

3 t=Turtle()

4 t.screen.bgcolor("black")

5 colors=["blue","purple","red","yellow"]

6 t.screen.tracer(0,0)

7

8 for x in range(300):

9 t.color(colors[x%4])

10 t.fd(x)

11 t.left(90)

12

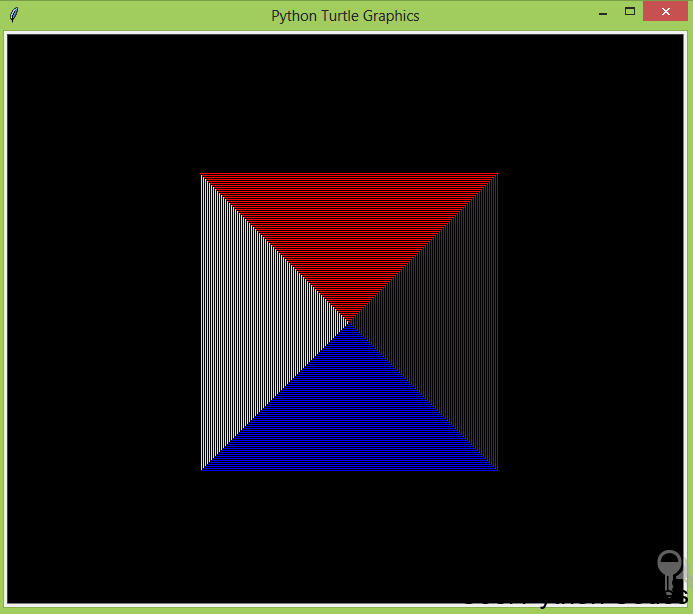
13 t.screen.exitonclick()

14 t.screen.mainloop()

Here’s the output of the code snippet above.

Beautiful Square

The below code snippet is saved as **beautiful\_spiral.py**



Python

1 from turtle import Turtle

2

3 t=Turtle()

4 t.screen.bgcolor("black")

5 colors=["blue","purple","red","yellow","orange","brown"]

6 t.screen.tracer(0,0)

7

8 for x in range(300):

9 t.color(colors[x%6])

10 t.fd(x)

11 t.left(59)

12

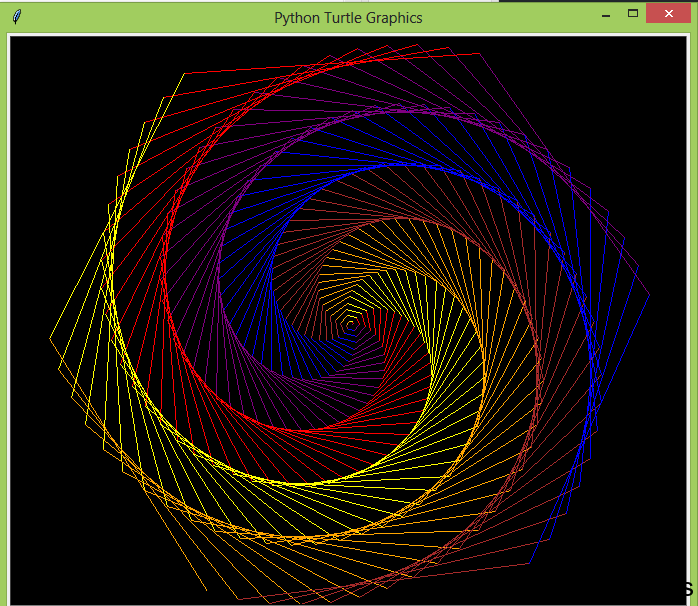
13 t.screen.exitonclick()

14 t.screen.mainloop()

Here’s the output of the above code snippet.

beautiful spiral

You have come to the end of this tutorial. Please don’t forget to:



Please if you have any questions or contributions, please leave it in the comment section.

Thanks for reading. If you enjoyed this tutorial, hit the share button. It would mean a lot to me and it helps other people to see the tutorial.