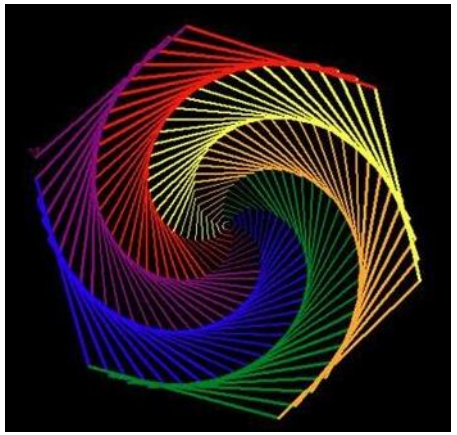
Module 3

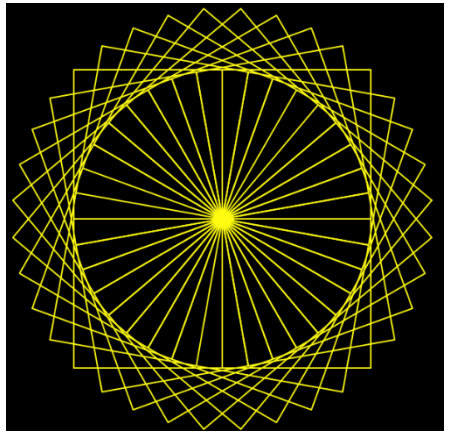
Module -3 (Graphics)

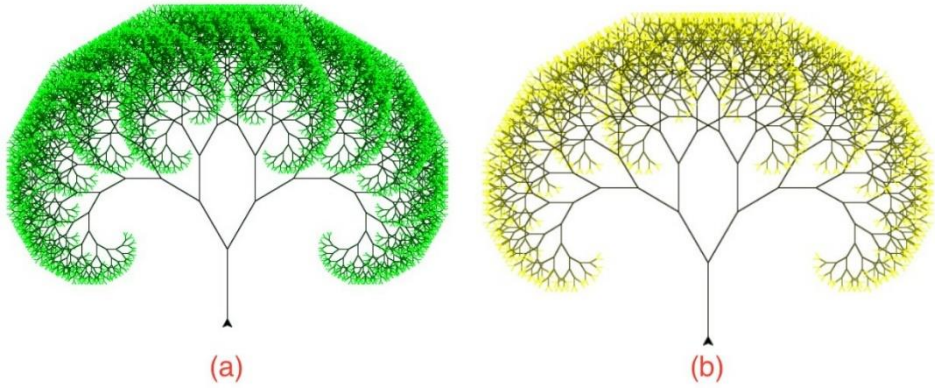
- Graphics – Terminal-based programs, Simple Graphics using Turtle, Operations, 2D Shapes, Colors and RGB Systems, A case study. Image Processing – Basic image processing with inbuilt functions.
- Graphical User Interfaces – Event-driven programming, Coding simple GUI-based programs : Windows, Labels, Displaying images, Input text entry, Popup dialog boxes, Command buttons, A case study.

What is Python with Turtle?

- Python with Turtle is a Python library that enables you to create virtual drawings and shapes. The "turtle" is a the pen that you use to draw.

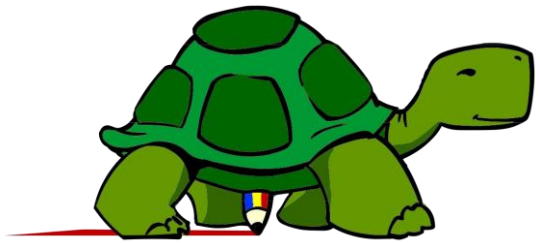




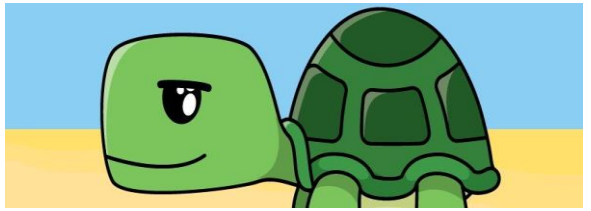


History

- Part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966
- Turtles are a type of educational robots designed in the 1940s, that are typically designed low to the ground. Some turtles are specifically designed to have pens attached to the back of them so they can draw on large sheets of paper.
- It uses tkinter for the underlying graphics, it needs a version of Python installed with Tk support.



A Basic Program on Turtle



- There is no programming language called ‘turtle’, so we use Python whenever we import the turtle library. Because of this, on our first line of code, we need to tell our computer we are using turtle:
  - >> import turtle

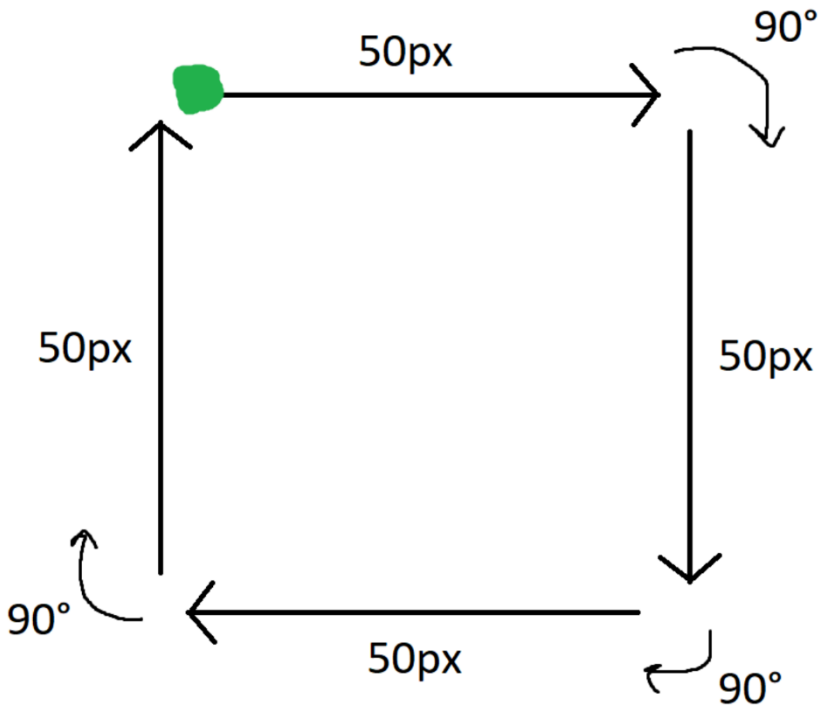
- Next, we need to name our turtle, and turtle will respond to whatever name you give it. We are just going to name it “alfred”.
  - >>alfred = turtle.Turtle()

Basic Turtle command

- There are four basic turtle commands
- turtle.forward(x)
  - Moves turtle forward in direction it is facing by x steps (right by default).
- turtle.back(x)
  - Moves turtle backward from its facing direction by x steps
- turtle.left(x)
  - Turns the turtle x degrees counterclockwise
- turtle.right(x)
  - Turns the turtle x degrees clockwise

Draw a square

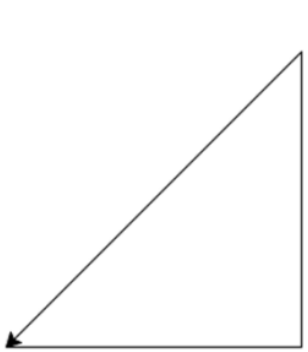
- let's try and draw a square. Here's a diagram showing what we'll need in order to accomplish that.

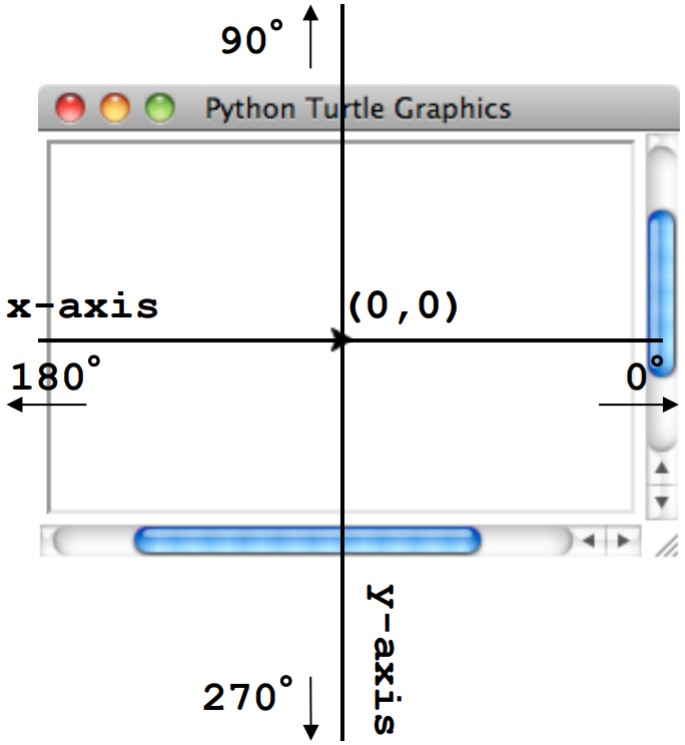



```
forward(50)
right(90)
forward(50)
right(90)
forward(50)
right(90)
forward(50)
```

Right-angled triangle

- We are going to draw a right-angled triangle



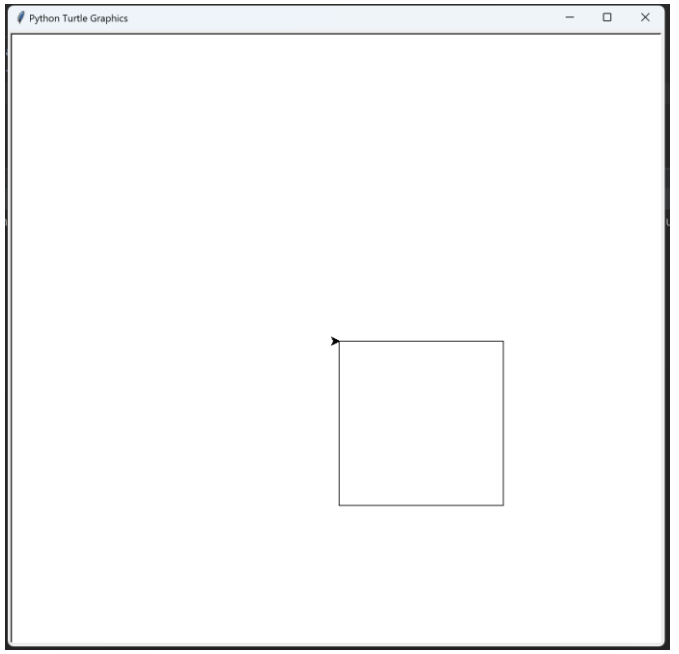



```
import math
import turtle
board = turtle.Turtle()
board.forward(100)  # draw base
board.left(90)
board.forward(100)
board.left(135)
x= math.sqrt(100**2+100**2) #C2 = a2 + b2
board.forward(x)
turtle.done()
```

Draw a square using a loop

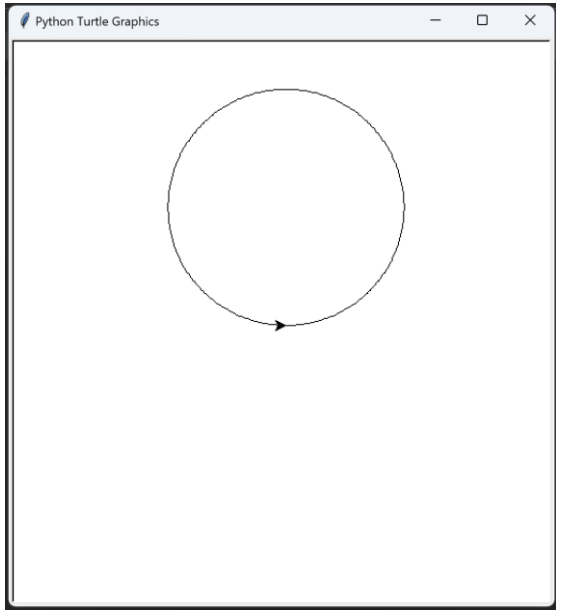
- We can use loops when drawing shapes using Turtle graphics
- Write a program that will draw a square using a loop

```
import turtle
board = turtle.Turtle()
count=0
while(count<4):
    board.forward(200)
    board.right(90)
    count+=1
turtle.done()
```



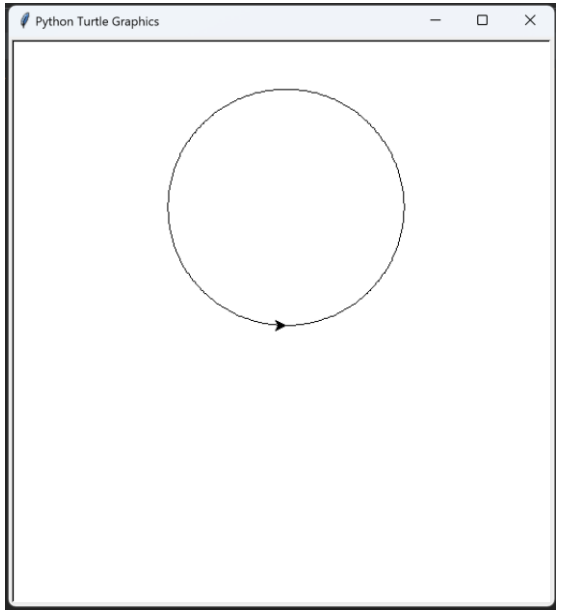
Draw a circle

- Write a program that will draw a circle
  - Steps:
  - Draw a short line (2 pixels)
  - Turn 1 degree
  - Repeat the above steps 360 times



Draw a circle

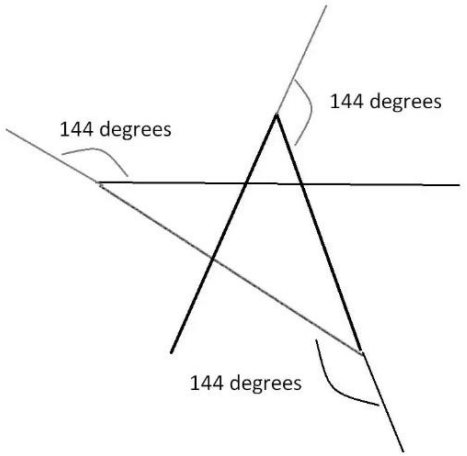
- Write a program that will draw a circle
  - Steps:
  - Draw a short line (2 pixels)
  - Turn 1 degree
  - Repeat the above steps 360 times




```
import turtle
count = 0
while(count < 360):
    turtle.forward(2)
    turtle.left(1)
    count = count + 1
print("Finished!")
turtle.done()
```

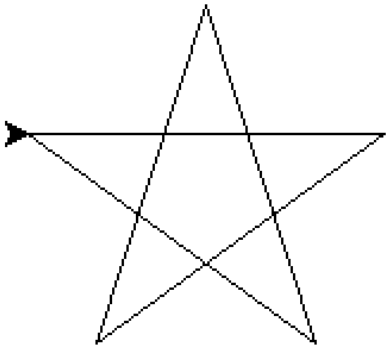
Star using Turtle In Python

- Import a turtle and create an object from it.
- Iterate the loop five times if you need five edges in the star.
- Move forward turtle x length and turn it 144 degrees



Drawing Stars with Turtle

```
import turtle
star = turtle.Turtle()
for i in range(5):
    star.forward(150)
    star.right(144)
turtle.done()
```



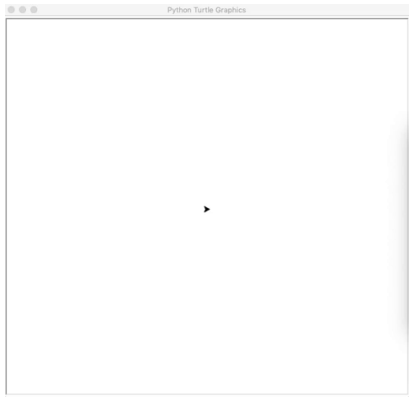
Draw Hexagon Using Turtle Graphics

- Define an instance for turtle.
- For a hexagon execute a loop 6 times.
- In every iteration move turtle 90 units forward and move it left 300 degrees.
- This will make up Hexagon .

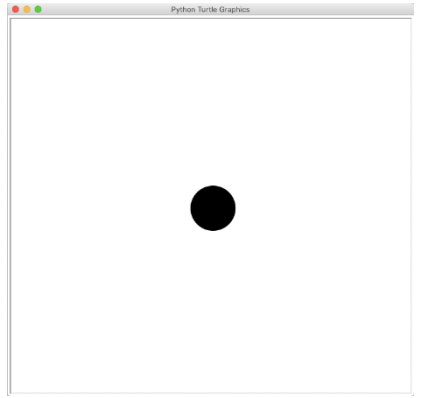
```
import turtle
hexagon = turtle.Turtle()
# executing loop 6 times for 6 sides
for i in range(6):
      # Move forward by 90 units
       hexagon.forward(90)
      # Turn left the turtle by 300 degrees
       hexagon.left(300)
```

Drawing Preset Figures

- You can use a single command to draw a circle:
  - `>>> t.circle(60)`

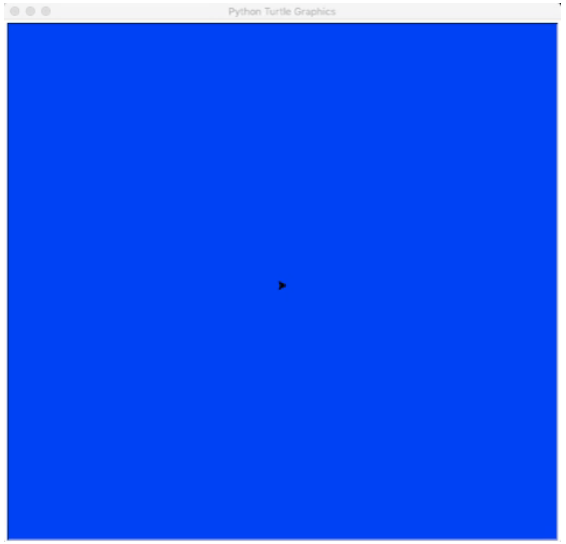


- In the same way, you can also draw a dot, which is nothing but a filled-in circle. Type in this command:
  - `>>> t.dot(20)`



Changing the Screen Color

- By default, turtle always opens up a screen with a white background. However, you can change the color of the screen at any time using the following command:
  - `>>> turtle.bgcolor("blue")`
- You can replace "blue" with any other color. Try "green" or "red". You'll get a result like this:



Changing the Screen Title

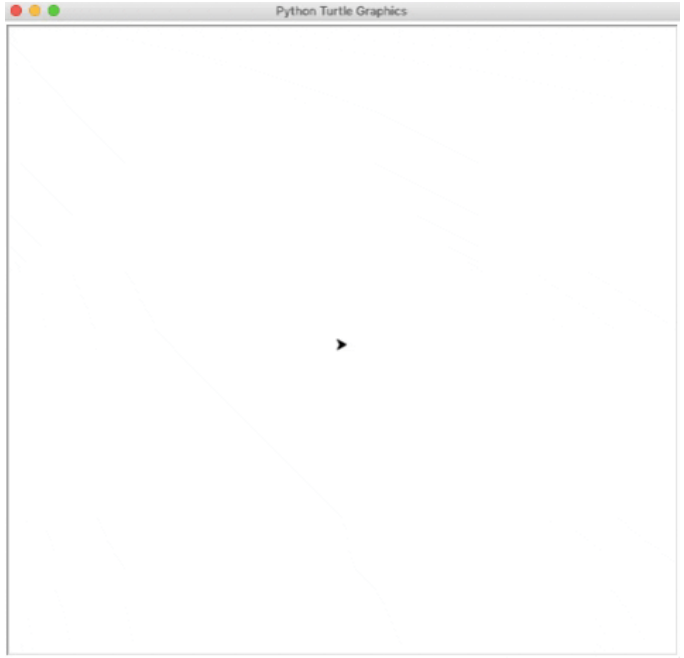
- Sometimes, you may want to change the title of your screen. You can change the title of your screen with the help of this command:
  - `turtle.title("My Turtle Program")`

Changing the Turtle Size

- You can increase or decrease the size of the onscreen turtle to make it bigger or smaller.
  - `>>> t.shapesize(1,5,10)`
  - `>>> t.shapesize(10,5,1)`
  - `>>> t.shapesize(1,10,5)`
  - `>>> t.shapesize(10,1,5)`
- The numbers given are the parameters for the size of the turtle:
  - Stretch length
  - Stretch width
  - Outline width

Changing the Pen Size

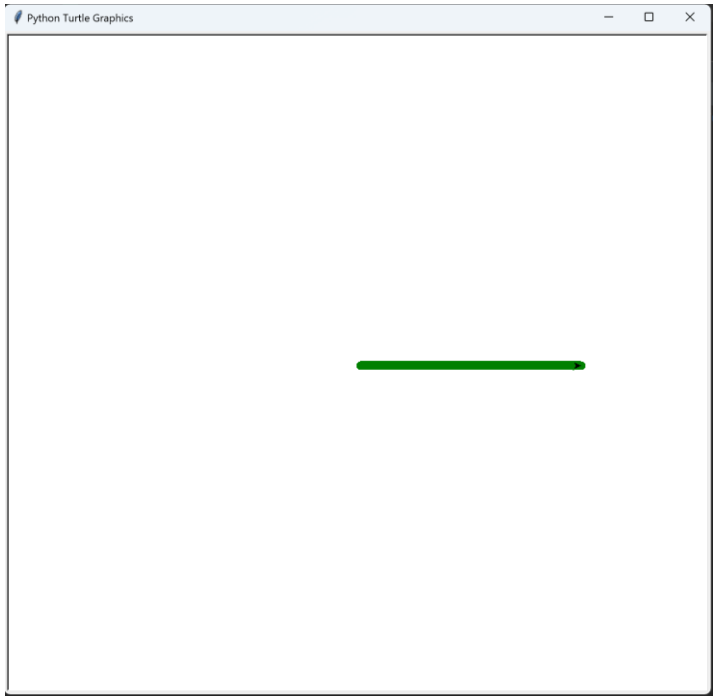
- you may need to increase or decrease the thickness of your pen. You can do this using the following command:
  - `>>> t.pensize(5)`
  - `>>> t.forward(100)`



Change the color of the pen

- To change the color of the pen (or the outline), you type the following:

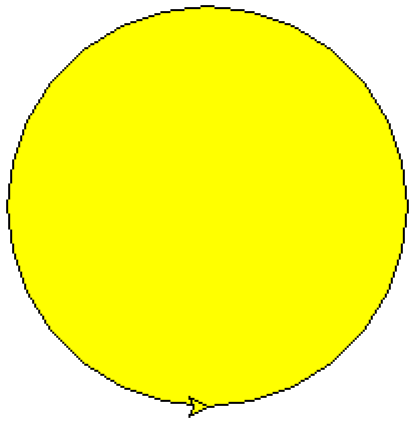
```
# import turtle library
import turtle
t=turtle.Turtle()
t.pensize(10)
t.pencolor("Green")
t.fd(250)
turtle.done()
```



Filling in an Image

- Coloring in an image usually makes it look better, doesn't it?
- The Python turtle library gives you the option to add color to your drawings.
- Try typing in the following code and see what happens:

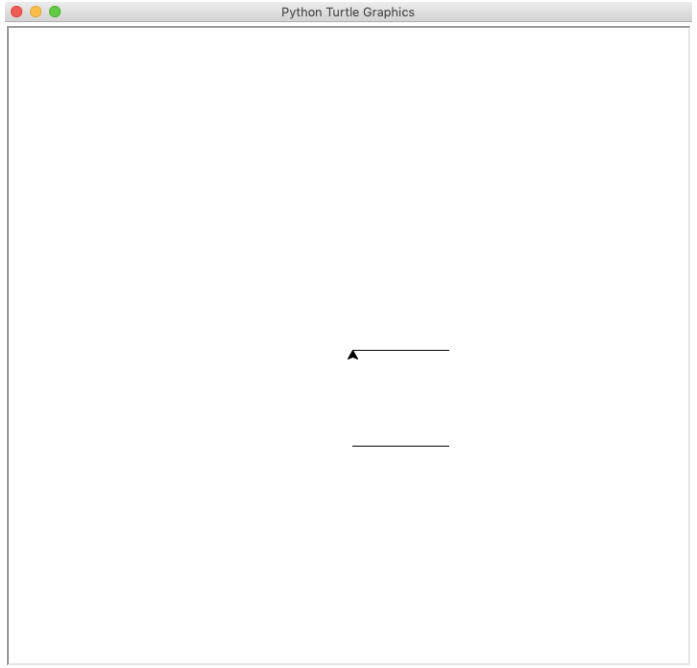
```
# import turtle library
import turtle
t=turtle.Turtle()
t.fillcolor("Yellow")
t.begin_fill()
t.circle(100)
t.end_fill()
turtle.done()
```



Picking the Pen Up and Down

- Sometimes, you may want to move your turtle to another point on the screen without drawing anything on the screen itself. To do this, you use `.penup()`
- Then, when you want to start drawing again, you use `.pendown()`

```
>>> t.fd(100)
>>> t.rt(90)
>>> t.penup()
>>> t.fd(100)
>>> t.rt(90)
>>> t.pendown()
>>> t.fd(100)
>>> t.rt(90)
>>> t.penup()
>>> t.fd(100)
>>> t.pendown()
```

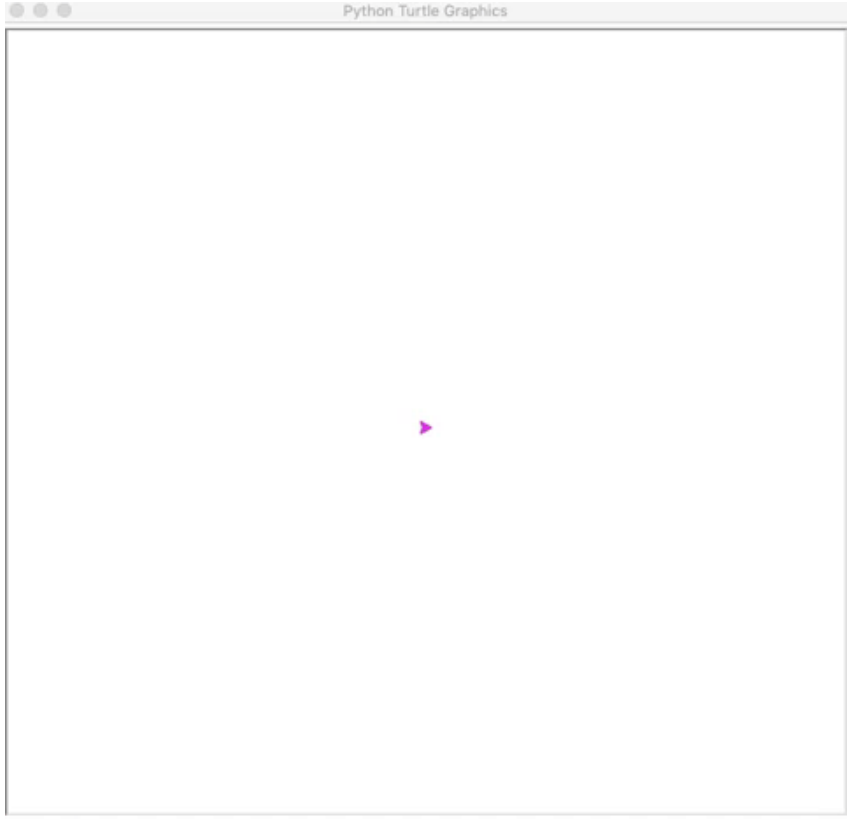


Here, you've obtained two parallel lines instead of a square by adding some extra commands in between the original program.

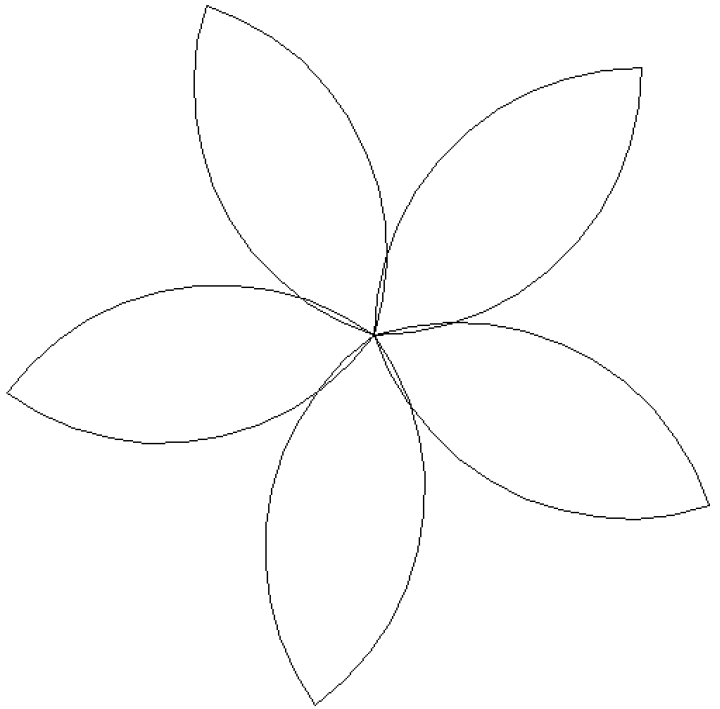
Cloning Your Turtle

- Sometimes, you may need to have more than one turtle on your screen.
- you can get another turtle by cloning your current turtle into your environment.

```
# import turtle library
import turtle
t=turtle.Turtle()
c = t.clone()
t.color("magenta")
c.color("red")
t.circle(100)
c.circle(60)
turtle.done()
```

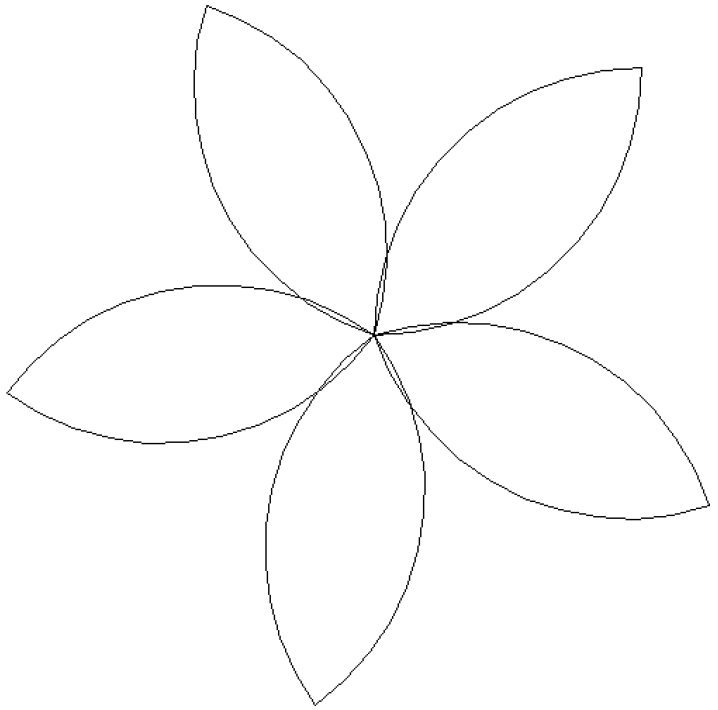


Draw a flower using Turtle



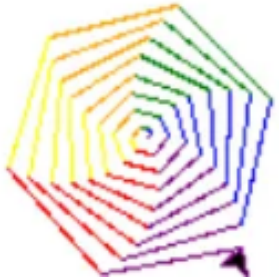
Draw a flower using Turtle

```
import turtle
t=turtle.Turtle()
t.speed(0)
def flower():
    for i in range(5):
        t.circle(190,90)
        t.left(90)
        t.circle(190,90)
        t.left(18)
flower()
turtle.done()
```



Questions

- Draw Spiral Helix Pattern
- Create `Rainbow Benzene`



- `Draw a red heart`

Spiral Helix

```
import turtle

new = turtle.Screen()
turtle.speed(2)

for i in range(10):
    turtle.circle(5*i)
    turtle.circle(-5*i)
```

Rainbow

```
import turtle
colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow']
t = turtle.Pen()
for x in range(50):
    t.pencolor(colors[x%6])
    t.forward(x)
    t.left(59)
```

Red Heart

```
import turtle
wn = turtle.Screen()
red = turtle.Turtle()

def curve():
    for i in range(200):
        red.right(1)
        red.forward(1)

def heart():
    red.fillcolor('red')
    red.begin_fill()
    red.left(140)
    red.forward(113)
    curve()
    red.left(120)
    curve()
    red.forward(112)
    red.end_fill()

heart()
turtle.done()
```

Image Processing

Image Processing

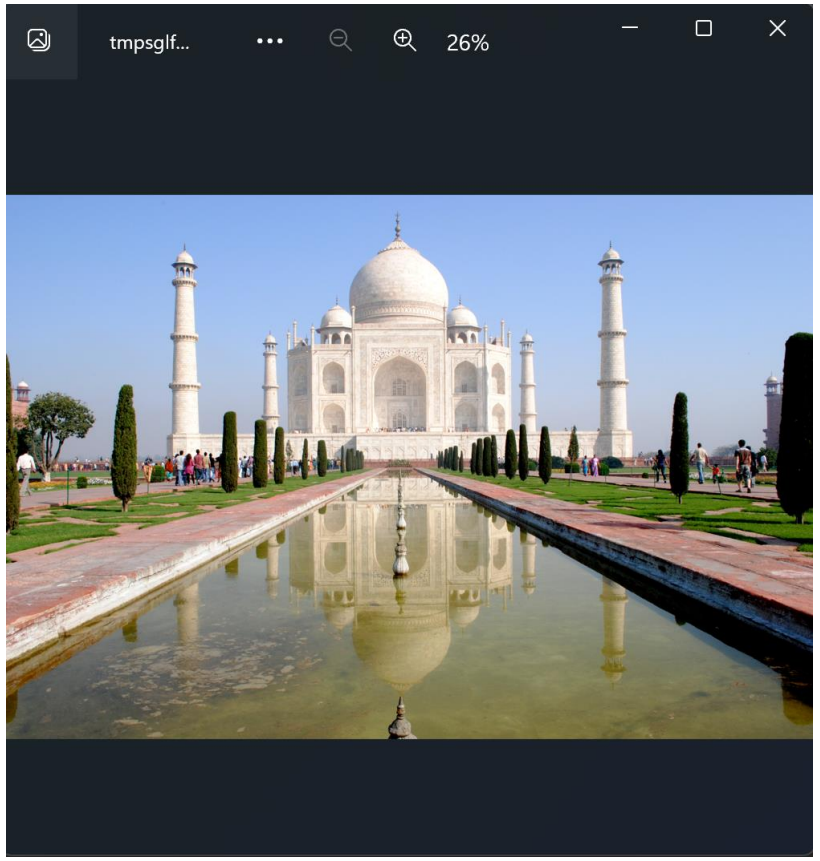
- Python provides lots of libraries for image processing, including
- **OpenCV** − Image processing library mainly focused on real-time computer vision with application in wide-range of areas like 2D and 3D feature toolkits, facial & gesture recognition, Human-computer interaction, Mobile robotics, Object identification and others.
- **Numpy and Scipy libraries** − For image manipuation and processing.
- **Sckikit** − Provides lots of alogrithms for image processing.
- **Python Imaging Library (PIL)** − To perform basic operations on images like create thumnails, resize, rotation, convert between different file formats etc.

Image: Open() and show()

#Import required library
from PIL import Image

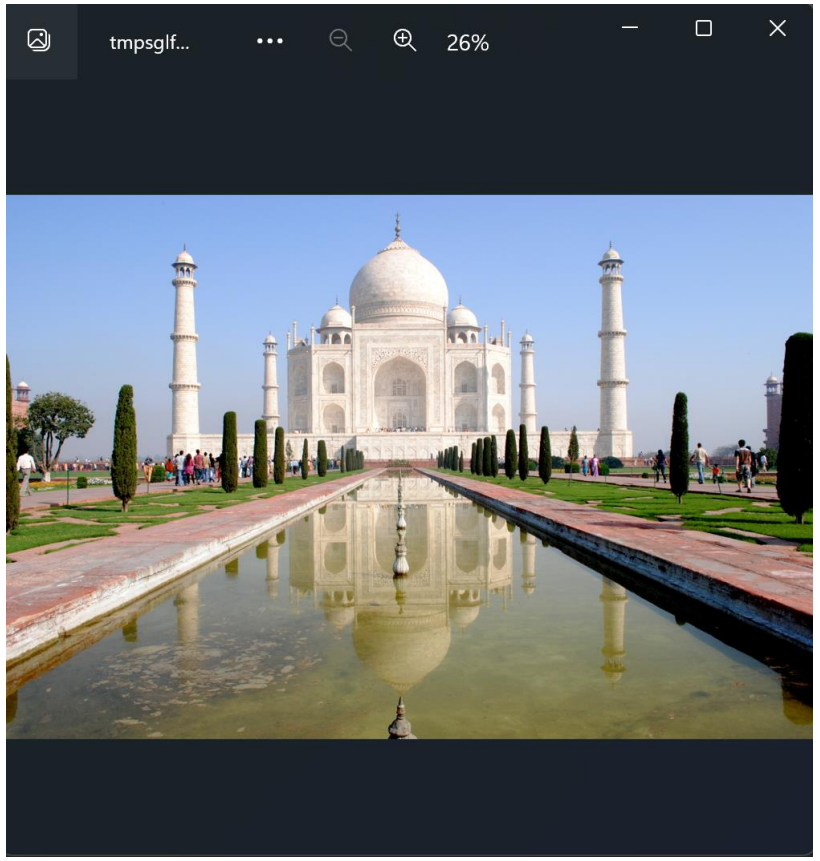
#Open Image
im = Image.open("Taj_Mahal_Front.jpg")

#Image show
im.show()



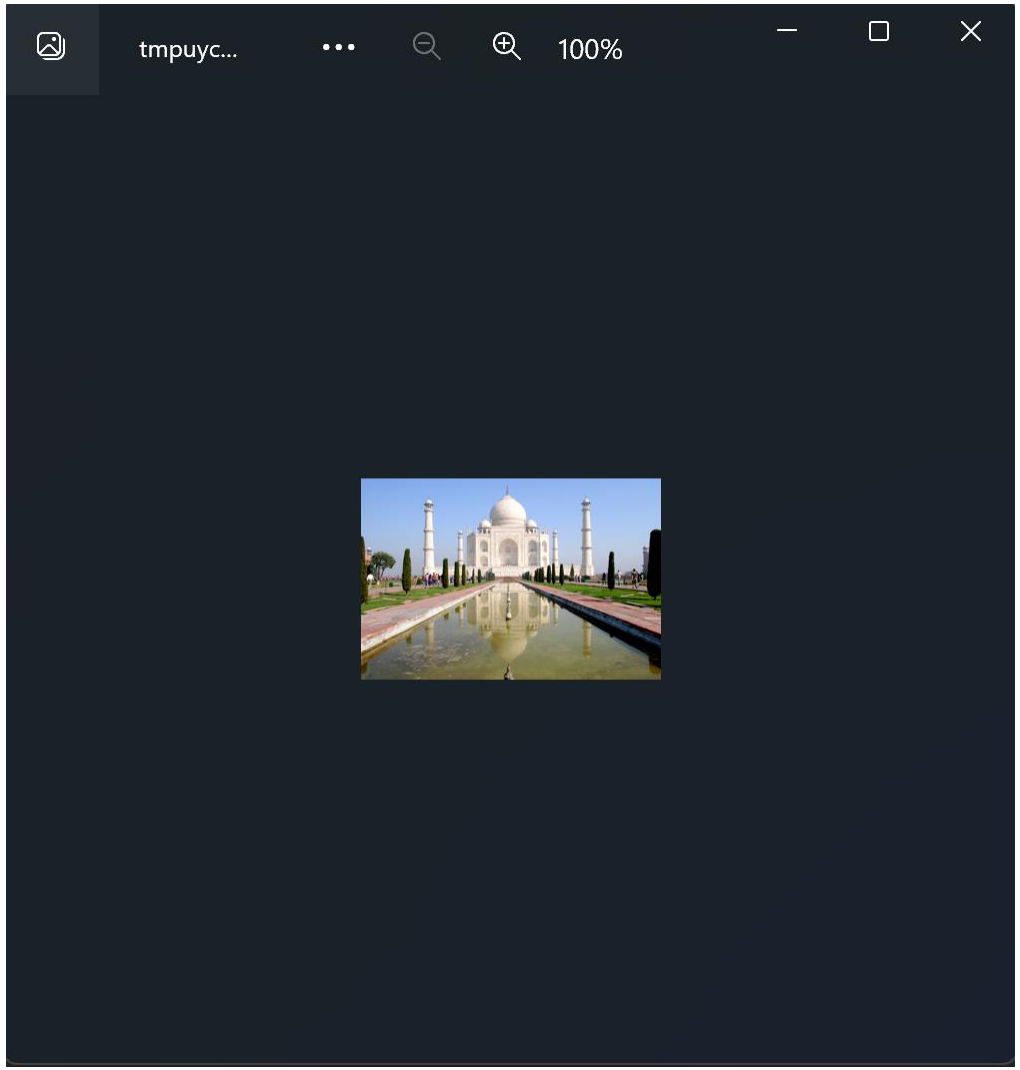
Convert and Save() Image

```
#Import required library
from PIL import Image

#Open Image
im = Image.open("Taj_Mahal_Front.jpg")

im.save('TajMahal.png')
```



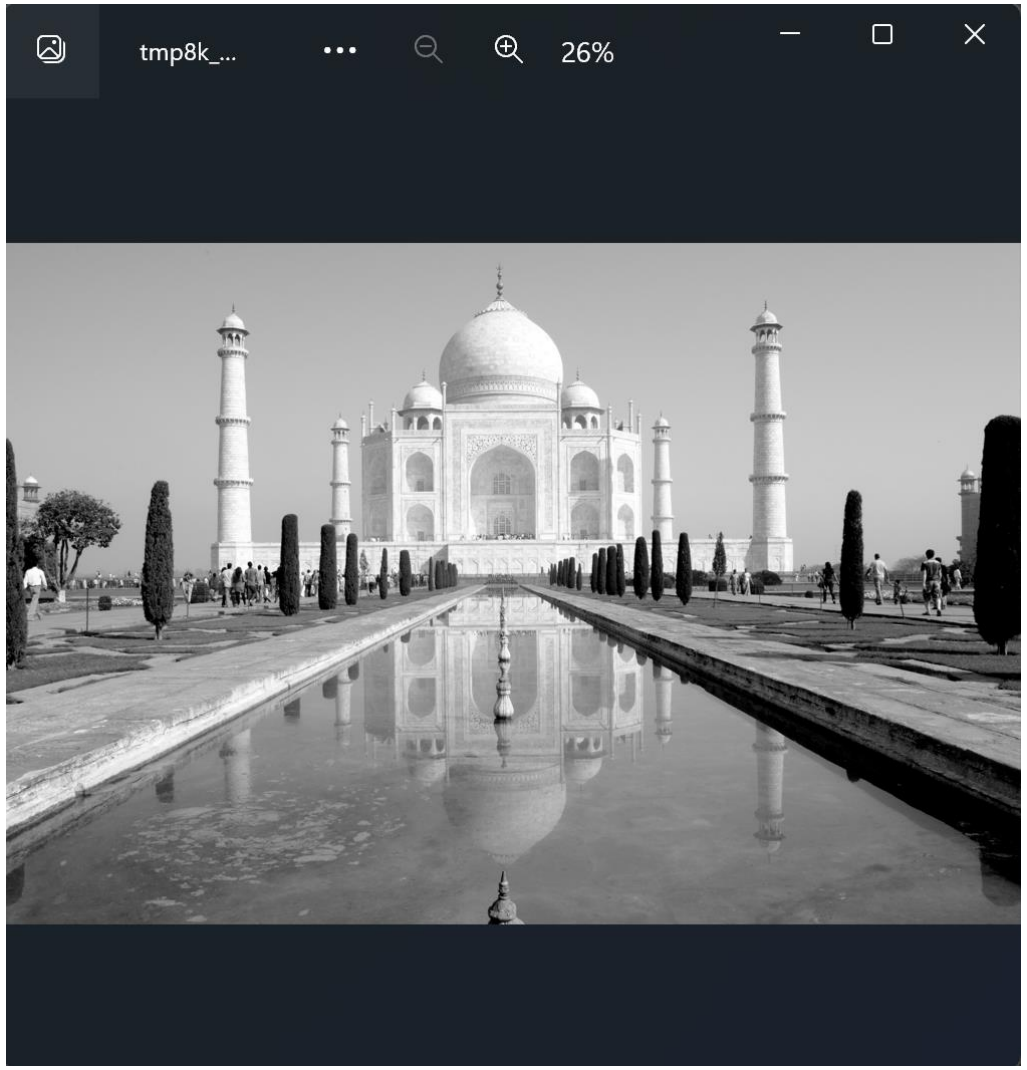
Resize-thumbnails()

```
#Import required library
from PIL import Image

#Open Image
im = Image.open("Taj_Mahal_Front.jpg")

im.thumbnail ((300, 300))
im.show()
```



Converting to grayscale image – convert()

```
#Import required library
from PIL import Image

#Open Image
im = Image.open("Taj_Mahal_Front.jpg")

TajMahal_gray = Image.open('
Taj_Mahal_Front.jpg ').convert('L')

TajMahal_gray.show()
```



References

- https://www.tutorialspoint.com/image-processing-in-python

Graphical User Interfaces

Python - GUI Programming (Tkinter)

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below:

- **Tkinter:** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this tutorial.
- **wxPython:** This is an open-source Python interface for wxWindows http://wxpython.org.
- **JPython:** JPython is a Python port for Java, which gives Python scripts seamless access to Java class libraries on the local machine http://www.jython.org.

Tkinter Programming:

- Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps:
  - **Example:**Import the *Tkinter* module.
  - Create the GUI application main window.
  - Add one or more of the above mentioned widgets to the GUI application.
  - Enter the main event loop to take action against each event triggered by the user.

```
import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

Python Tkinter Geometry

- The Tkinter geometry specifies the method by using which, the widgets are represented on display. The python Tkinter provides the following geometry methods.
  - pack() method
  - grid() method
  - place() method

Python Tkinter Geometry - pack()

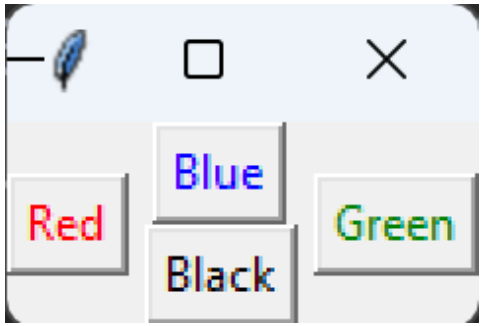
- The pack() widget is used to organize widget in the block.

```
widget.pack( pack_options )
```

- Here is the list of possible options −
  - **expand** − When set to true, widget expands to fill any space not otherwise used in widget's parent.
  - **fill** − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
  - **side** − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

Python Tkinter Geometry - pack()

```
from tkinter import *
parent = Tk()
redbutton = Button(parent, text = "Red", fg = "red")
redbutton.pack( side = LEFT)
greenbutton = Button(parent, text = "Green", fg = "green")
greenbutton.pack( side = RIGHT )
bluebutton = Button(parent, text = "Blue", fg = "blue")
bluebutton.pack( side = TOP )
blackbutton = Button(parent, text = "Black", fg = "black")
blackbutton.pack( side = BOTTOM)
parent.mainloop()
```



Python Tkinter grid() method

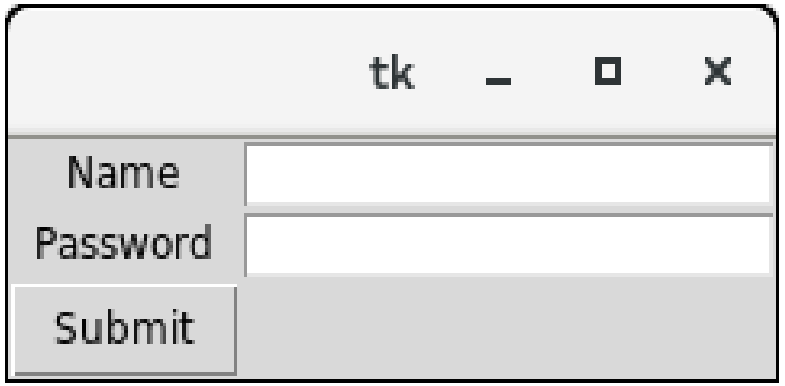
- The grid() geometry manager organizes the widgets in the tabular form.
- We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or rowspan(height) of a widget.

```
widget.grid(options)
```

- Column
  - The column number in which the widget is to be placed. The leftmost column is represented by 0.
- Columnspan
  - The width of the widget. It represents the number of columns up to which, the column is expanded.
- row
  - The row number in which the widget is to be placed. The topmost row is represented by 0.
- rowspan
  - The height of the widget, i.e. the number of the row up to which the widget is expanded.

Python Tkinter grid() method

```
from tkinter import *
parent = Tk()
name = Label(parent,text = "Name").grid(row = 0, column = 0)
e1 = Entry(parent).grid(row = 0, column = 1)
password = Label(parent,text = "Password").grid(row = 1, column = 0)
e2 = Entry(parent).grid(row = 1, column = 1)
submit = Button(parent, text = "Submit").grid(row = 4, column = 0)
parent.mainloop()
```



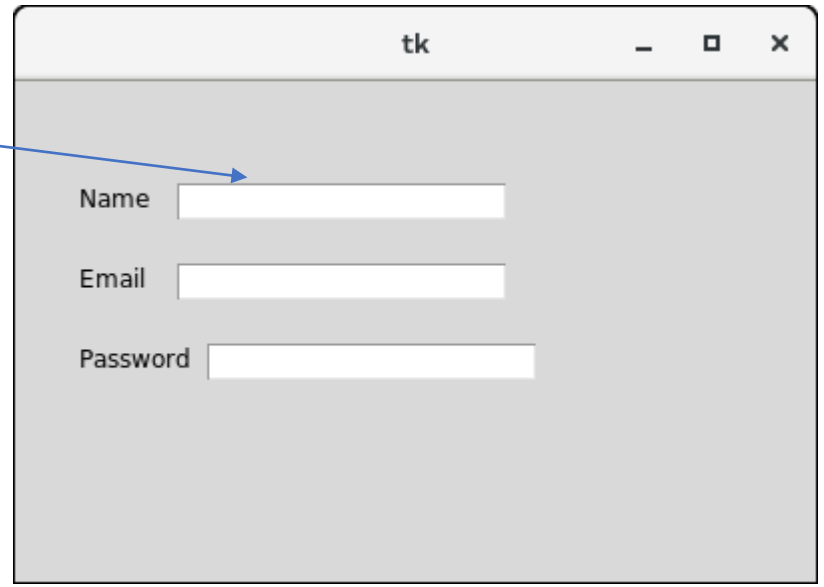
Python Tkinter place() method

- The place() geometry manager organizes the widgets to the specific x and y coordinates.

```
widget.place(options)
```

Python Tkinter place() method

```
from tkinter import *
top = Tk()
top.geometry("400x250")
name = Label(top, text = "Name").place(x = 30,y = 50)
email = Label(top, text = "Email").place(x = 30, y = 90)
password = Label(top, text = "Password").place(x = 30, y = 130)
e1 = Entry(top).place(x = 80, y = 50)
e2 = Entry(top).place(x = 80, y = 90)
e3 = Entry(top).place(x = 95, y = 130)
top.mainloop()
```



Python Tkinter – MessageBox Widget

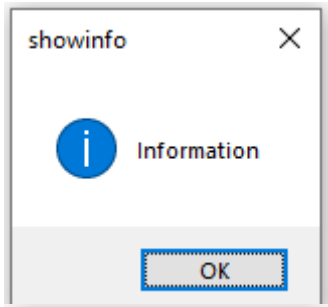
- MessageBox Widget is used to display the message boxes in the python applications.

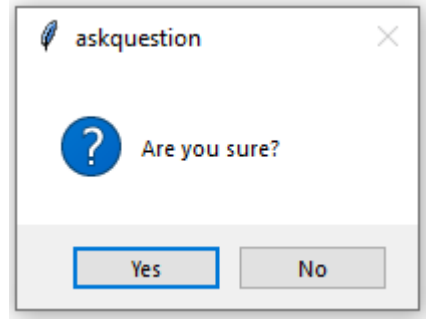
```
messagebox.Function_Name(title, message [, options])
```

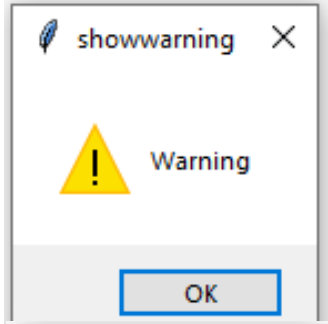
- Function_Name:
  - showinfo(): Show some relevant information to the user.
  - showwarning(): Display the warning to the user.
  - showerror(): Display the error message to the user.
  - askquestion(): Ask question and user has to answered in yes or no.
  - askokcancel(): Confirm the user's action regarding some application activity.
  - askyesno(): User can answer in yes or no for some action.
  - askretrycancel(): Ask the user about doing a particular task again or not.

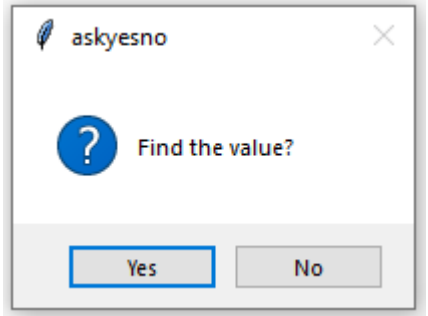
Python Tkinter – MessageBox Widget

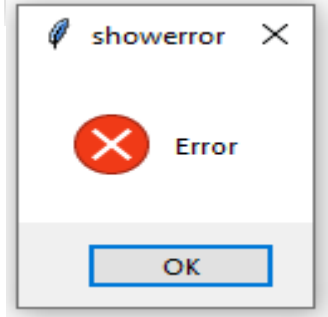
```
from tkinter import *
from tkinter import messagebox
root = Tk()
root.geometry("300x200")

w = Label(root, text ='Messagebox Demo', font = "50")
w.pack()

messagebox.showinfo("showinfo", "Information")
messagebox.showwarning("showwarning", "Warning")
messagebox.showerror("showerror", "Error")
messagebox.askquestion("askquestion", "Are you sure?")
messagebox.askokcancel("askokcancel", "Want to continue?")
messagebox.askyesno("askyesno", "Find the value?")
messagebox.askretrycancel("askretrycancel", "Try again?")

root.mainloop()
```

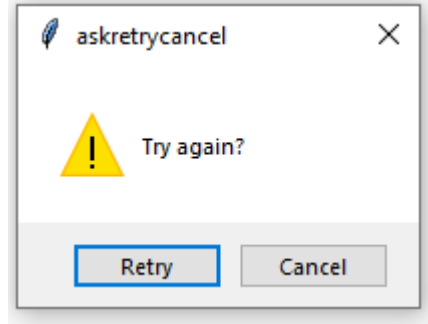












Python - Tkinter Button

Python - Tkinter Button

The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button, which is called automatically when you click the button.

Syntax:

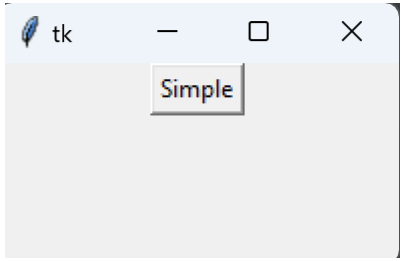
```
w = Button ( master, option=value, ... )
```

Parameters:

- **master:** This represents the parent window.
- **options:** Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Python - Tkinter Button

```
from tkinter import *
top = Tk()
top.geometry("200x100")
b = Button(top, text="Simple")
b.pack()
top.mainloop()
```

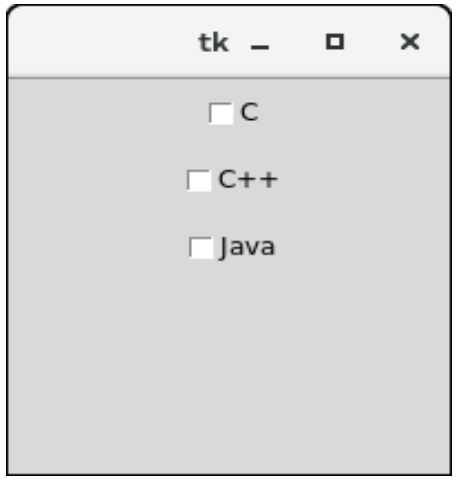


Python Tkinter Checkbutton

The Checkbutton is used to track the user's choices provided to the application. In other words, we can say that Checkbutton is used to implement the on/off selections.

```
w = checkbutton(master, options)
```

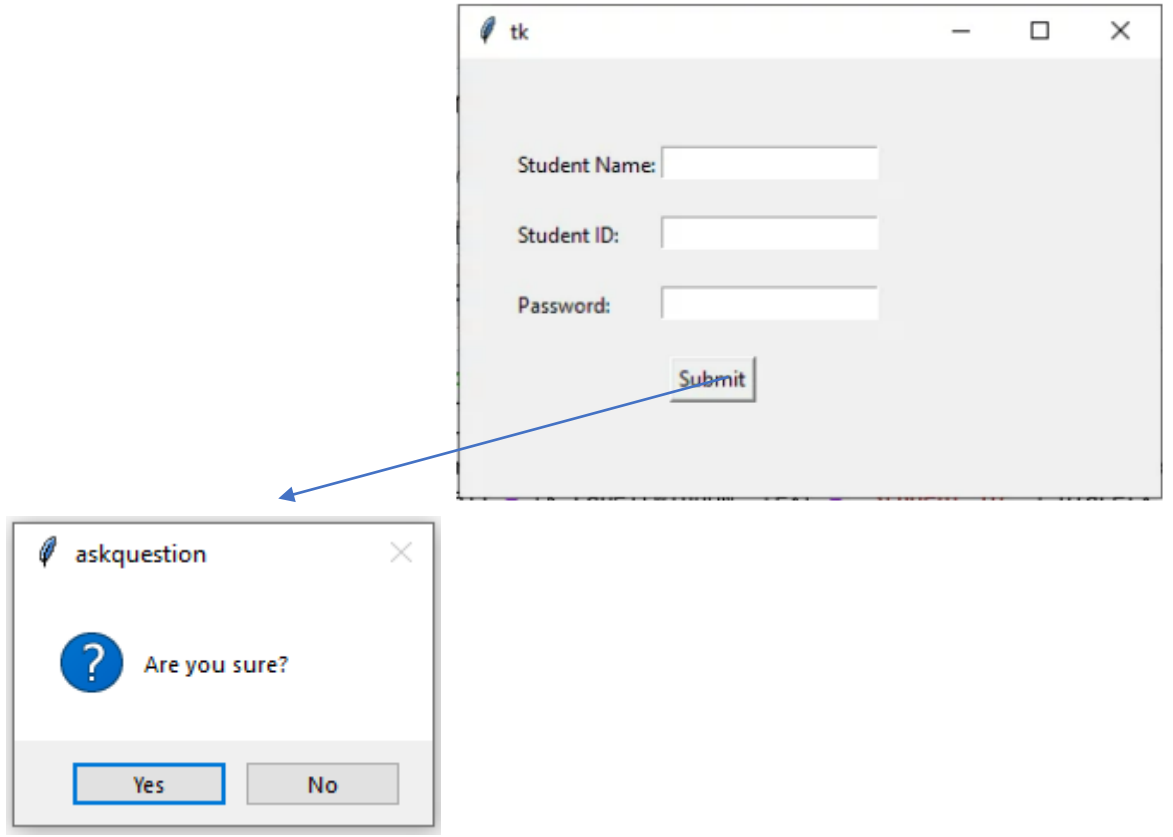
Python Tkinter Checkbutton




```
from tkinter import *
top = Tk()
top.geometry("200x200")
checkvar1 = IntVar()
checkvar2 = IntVar()
checkvar3 = IntVar()
chkbtn1 = Checkbutton(top, text="C", variable=checkvar1, onvalue=1, offvalue=0,
height=2, width=10)
chkbtn2 = Checkbutton(top, text="C++", variable=checkvar2, onvalue=1, offvalue=0,
height=2, width=10)
chkbtn3 = Checkbutton(top, text="Java", variable=checkvar3, onvalue=1, offvalue=0,
height=2, width=10)
chkbtn1.pack()
chkbtn2.pack()
chkbtn3.pack()
top.mainloop()
```

Python - Exercise

Write a python Tkinter program to display three text box in order to fill the information of student name, student id and password. After filling in all the information, the submit button must be appeared for submission purpose.



Python – Exercise - Solution

```
import tkinter as tk
window = tk.Tk()
window.geometry("400x250")
name = tk.Label(window, text = "Student Name:").place(x = 30, y = 50)
email = tk.Label(window, text = "Student ID:").place(x = 30, y = 90)
password =  tk.Label(window, text = "Password:").place(x = 30, y = 130)
sbmitbtn = tk.Button(window, text = "Submit", activebackground = "green",
activeforeground = "blue").place(x = 120, y = 170)
entry1 = tk.Entry(window).place(x = 115, y = 50)
entry2 = tk.Entry(window).place(x = 115, y = 90)
entry3 = tk.Entry(window).place(x = 115, y = 130)
window.mainloop()
```

References

- https://www.tutorialspoint.com/image-processing-in-python