M.Tech : System on Chip Design

Indian Institute of Technology, Palakkad

CS5107: Programming Lab

Lab 2: The Plot Thickens

Name: R. Parnika Murty

Roll No.: 152002016

(The assignment is done using Python3)

Version of python used:

```
Command Prompt

C:\Users\rpmur\Desktop\CS5107>python --version
Python 3.7.4

C:\Users\rpmur\Desktop\CS5107>
```

1) **Q2.py:** To create an object of PieChart type from a dictionary mapping labels to positive numbers. It also should throw exceptions when label is not a string and value is not a positive numeric.

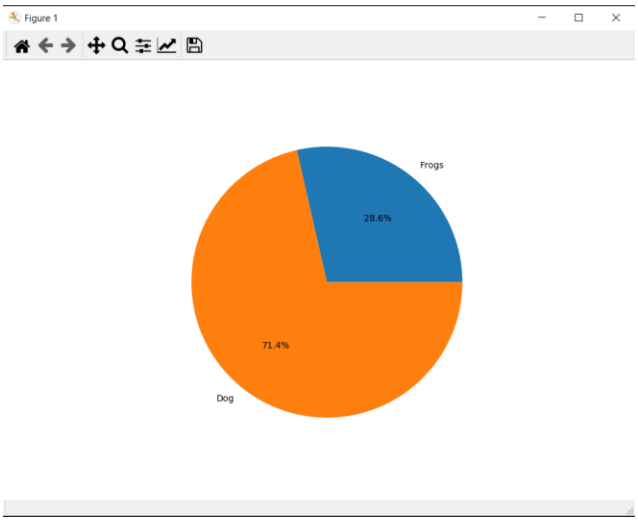
```
#to create a class PieChart
class PieChart:
    listkey=[]
    listval=[]
    listt=[]
    dict1={}
    #For the constructor part.
    def __init__(self,Dictt):
        self.dicts = Dictt
        for key in self.dicts:
            #setattr(self, key, Dictt[key])
            self.k = key
            self.v = self.dicts[key]
            #using lists to store values and labels to be able to display in piechart.
            self.listt.append([self.k,self.v])
            self.listkey.append(key)
            self.listval.append(self.dicts[key])
            #Exceptions as required.
            if type(self.k) != str :
                raise Exception("Label should be string")
            elif type(self.v) != int or self.v < 0 :
                raise Exception("Value should be a postive numeric")
```

The inputs:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    #p = p + ("Cat", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

The output:

```
Command Prompt

C:\Users\rpmur\Desktop\CS5107>cd 152002016_CodePy2_Parnika

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```



To test Exceptions:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":"10","Dog":25})
    #p = p + ("Cat", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

Command Prompt

```
C:\Users\rpmur\Desktop\CS5107>cd 152002016_CodePy2_Parnika

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

```python
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":-10,"Dog":25})
    #p = p + ("Cat", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

```
C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({1:10,"Dog":25})
    #p = p + ("Cat", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

```
value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Label should be string

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

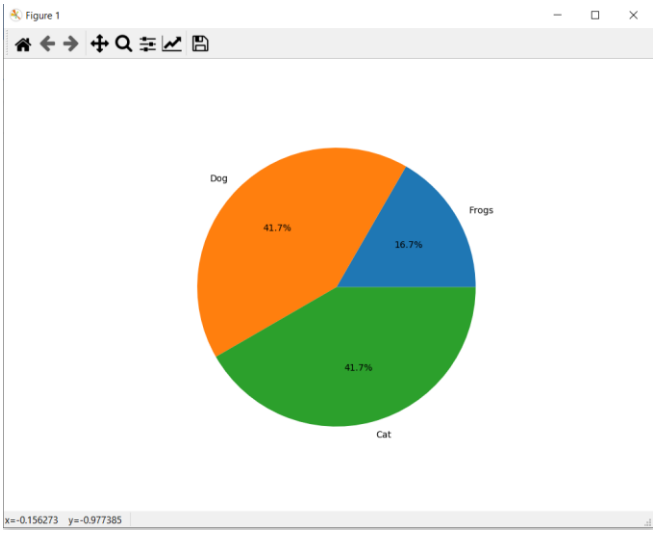
To add tuple:

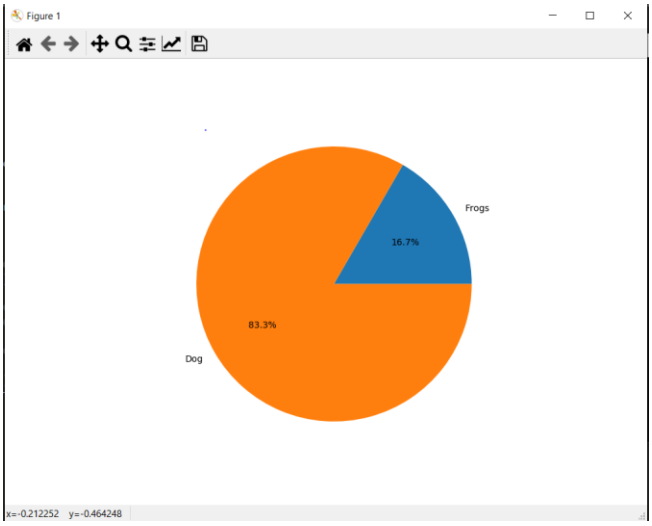
Input:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + ("Cat", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + ("Dog", 25)
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

Output:





To throw exception if any of the tuple parameters are not correct:

Input and outputs:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + ("Dog")
```

```
C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Tuple should be of length 2

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + ("Dog",-25)
```

```
C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + ("Dog","25")
```

```
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    p = p + (1,25)
```

```
Value should be a postive numeric

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>python Q2.py
<class 'Exception'>
Label should be string

C:\Users\rpmur\Desktop\CS5107\152002016_CodePy2_Parnika>
```

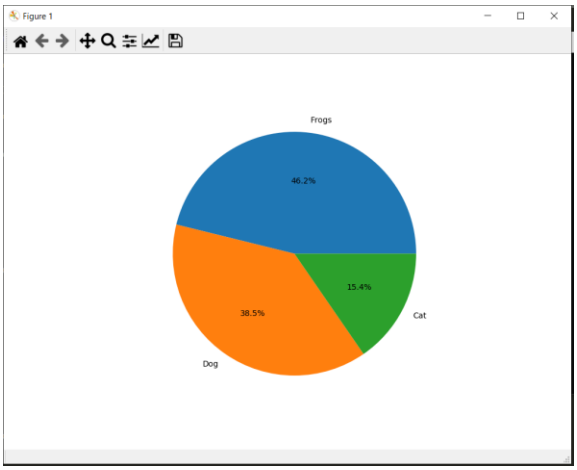
To add a PieChart object by operator overloading:

```
    def __add__(self,other):
        self.other = other
        if type(other) == tuple:
            self.listt.append([self.other[0],self.other[1]])
            if len(self.other) != 2 :
                raise Exception("Tuple should be of length 2")
            elif type(self.other[0]) != str :
                raise Exception("Label should be string")
            elif type(self.other[1]) != int or self.other[1] < 0 :
                raise Exception("Value should be a postive numeric")

        for pair in self.listt:
            if pair[0] not in self.dict1 :
                self.dict1[pair[0]] = pair[1]
            else:
                self.dict1[pair[0]] = self.dict1[pair[0]] + pair[1]
        self.listkey.clear()
        self.listval.clear()
        self.dicts = self.dict1
        for key in self.dict1:
            self.listkey.append(key)
            self.listval.append(self.dict1[key])
        #print(self.listkey)
        #print(self.listval)
        return self
```

Input:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    #p = p + ("Cat",25)
    p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = PieChart({"Frogs":10,"Dog":20})
    #p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    #p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

Output:



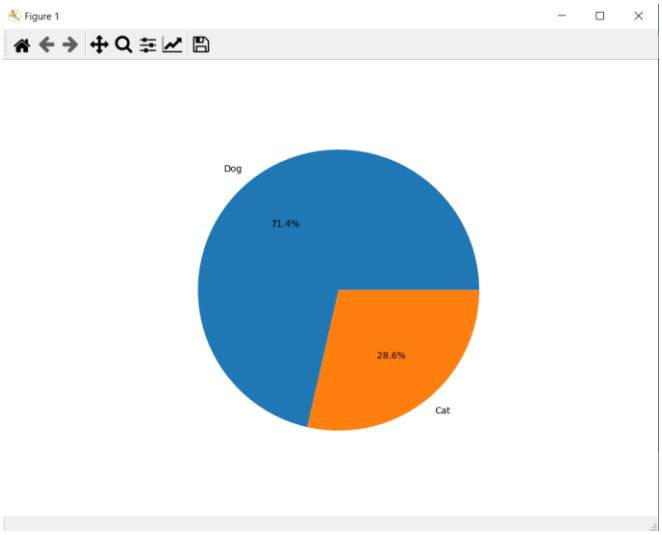
To Overload '–' operator:

```
    #Overloading subtract function.
    def __sub__(self,str1):
        if str1 in self.dicts:
            self.dicts.pop(str1)
        self.listkey.clear()
        self.listval.clear()
        for key in self.dicts:
            self.listkey.append(key)
            self.listval.append(self.dicts[key])
        for pair in self.listt:
            if pair[0] == str1:
                self.listt.remove(pair)
        return self
```

Input:

```
try:
    #p = PieChart({"Frogs":10,"Dog":20, "Cat":30})
    p = PieChart({"Frogs":10,"Dog":25})
    #p = p + ("Cat",25)
    p = p + PieChart({"Frogs":20,"Cat":10})
    #p = PieChart({"Frogs":10,"Dog":20})
    p = p - 'Frogs'
    #p = p + PieChart({"Frogs": 20, "Cat": 10})
    p = p - 'Lions'
    fig = plt.figure(figsize =(10, 7))
    plt.pie(p.listval, labels = p.listkey,autopct='%1.1f%%')
    plt.show()
except Exception as e:
    print(type(e))
    print(e)
```

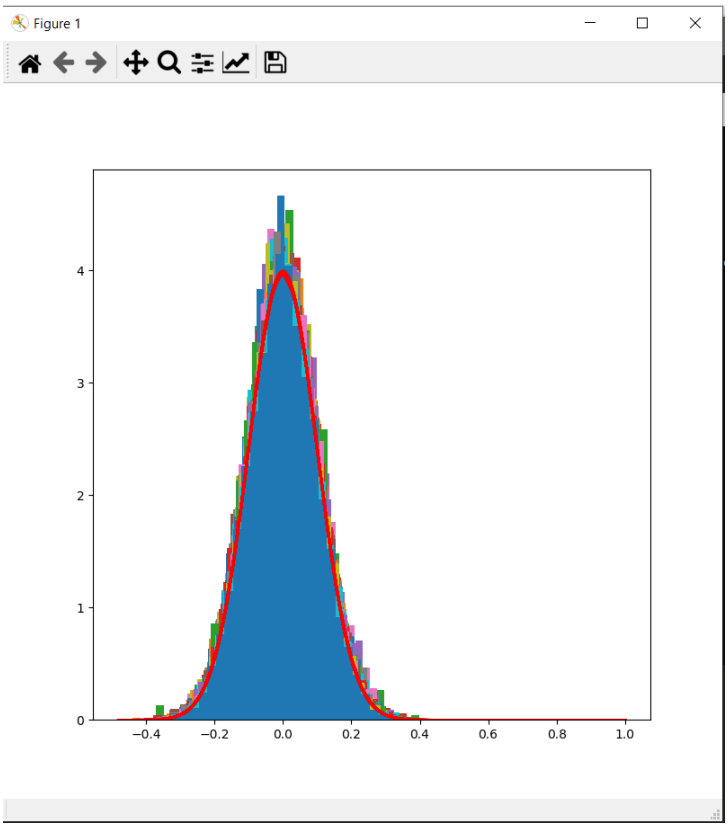
Output:



2) **Q3.py: To** demonstrate CLT using animation:

Q3.py

```python
#!/bin/python3
import matplotlib.animation as animation
import numpy as npy
import matplotlib.pyplot as plot


mean = 0
std = 0.1

fig, ax = plot.subplots(1, 1, figsize = (8, 8)) #pyplot.subplots creates a figure and a grid of subplots with a single call,
                                                # while providing reasonable control over how the individual plots are created.

#To start the animation using 'matplotlib.animation' to plot the graph how often animate(i) is being called.
def animate(i):
    array = npy.random.normal(0, 0.1, 1000*i)
    count, bins, ignored = plot.hist(array, 30, normed=True)
    plot.plot(bins, 1/(std * npy.sqrt(2 * npy.pi)) *
              npy.exp( - (bins - mean)**2 / (2 * std**2) ),
              linewidth=2, color='r')

#Makes an animation by repeatedly calling a function func.
#The figure object that is used to get draw, resize, and any other needed events. The function to call at each frame
ani = animation.FuncAnimation(fig, animate, frames = 11, interval = 200)
plot.show()

#Credits: https://docs.scipy.org/doc/numpy-1.13.0/reference/
    # generated/numpy-random-normal-1.py
```



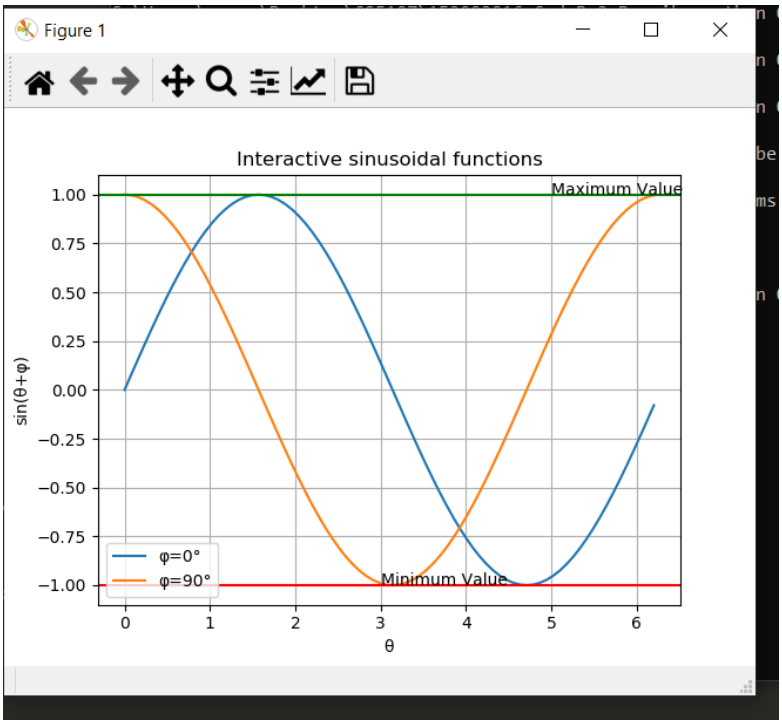
3) **Q4.py:** To create an object of type Sines and add offset to the sine curve:

Q4.py

```
class Sines:
    t = 0
    phi = 0
    deg = 0
    sr = 0
    sl =0
    def __init__(self): #Adding constructor
        self.phi = 0
        self.t = 0
        self.deg = 0
        self.sr = 0
        self.sl = 0

    def addSine(self,phi): #To add the offset and then calculating the radian of the degree.
        self.phi = phi
        self.deg = self.phi * (np.pi/180)
        #t = [0,30,45,60,90]
        #t = np.linspace(0,2*np.pi)
        #self.t = np.arange(0,2*np.pi,0.025*np.pi)
        #x = [i*(np.pi/180) for i in t]

    def show(self):
        self.t = np.arange(0,2*np.pi,0.025*np.pi)
        plt.plot(self.t,np.sin(self.t),label=chr(966)+"="+"0"+chr(176))
        #plt.plot(x,np.sin(x),marker='^',label="sin")
        if self.phi != 0:
            plt.plot(self.t,np.sin(self.t+self.deg),label=chr(966)+"="+str(self.phi)+chr(176))
        #plt.plot(x,np.cos(x),marker='+',label="cos")
        #plt.plot(self.t,np.sin(self.t+self.deg),label=chr(966)+"="+"90"+chr(176))
        #plt.xticks(t)
        plt.axhline(y=1, color='g', linestyle='-')
        plt.text(5,1,"Maximum Value")
        plt.axhline(y=-1, color='r', linestyle='-')
        plt.text(3,-1,"Minimum Value")
        plt.grid(True)
        plt.title("Interactive sinusoidal functions")
        plt.ylabel("sin"+"("+chr(952)+"+"+chr(966)+")")
        plt.xlabel(chr(952))
        plt.legend()
        plt.show()
```

```
s = Sines()
s.addSine(0)
s.addSine(90)
#s.shiftRight(45)
#s.shiftLeft(45)
s.show()
```

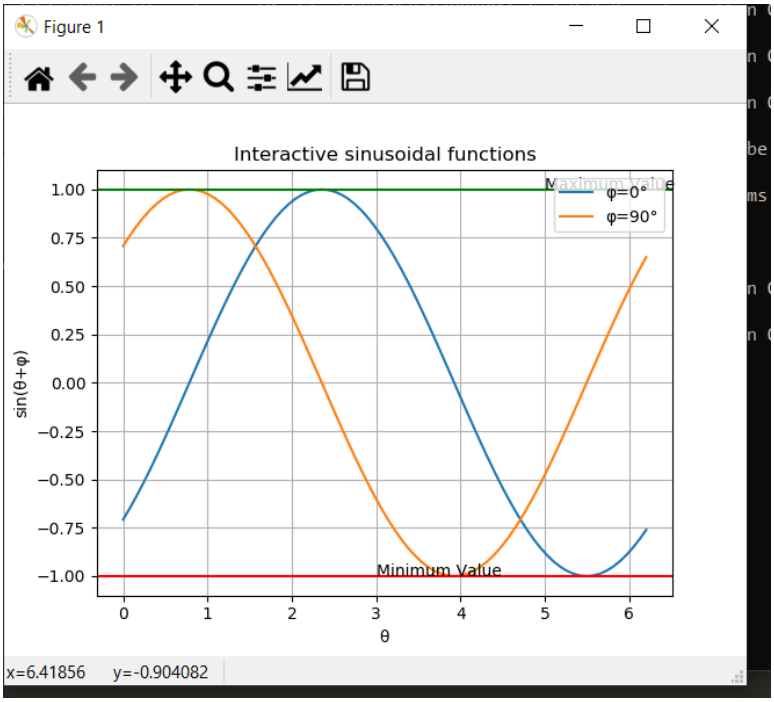
Output:



To shift right:

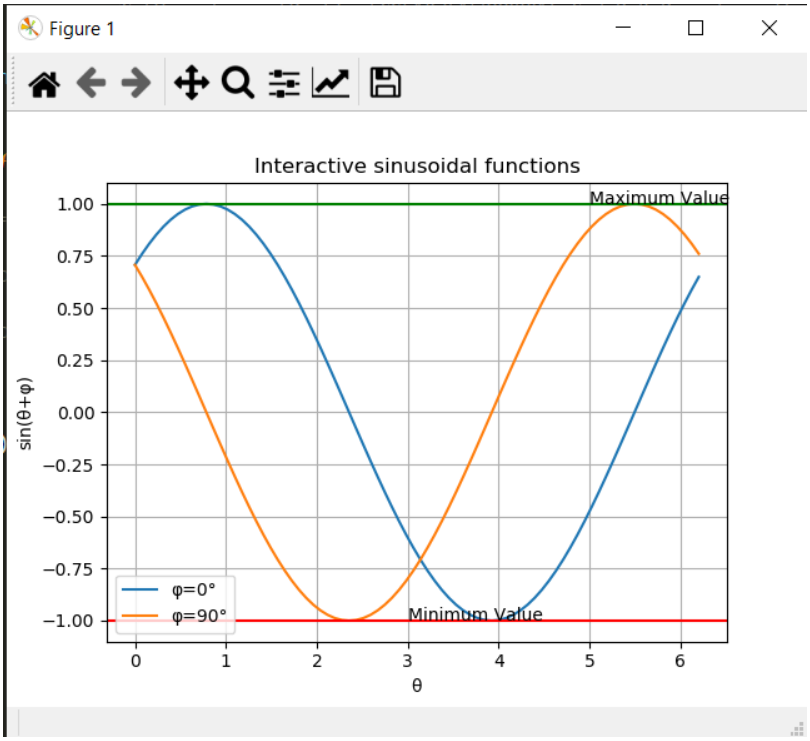
```
def shiftRight(self,sr):        #To shift graph right by the amount of offset mentioned while calling the method.
    self.sr = sr * (np.pi/180)
    self.t = np.arange(0,2*np.pi,0.025*np.pi)
    plt.plot(self.t,np.sin(self.t-self.sr),label=chr(966)+"="+"0"+chr(176))
    #plt.plot(x,np.sin(x),marker='^',label="sin")
    if self.phi != 0:
        plt.plot(self.t,np.sin(self.t+self.deg-self.sr),label=chr(966)+"="+str(self.phi)+chr(176))
    #plt.plot(x,np.cos(x),marker='+',label="cos")
    #plt.plot(self.t,np.sin(self.t+self.deg),label=chr(966)+"="+"90"+chr(176))
    #plt.xticks(t)
    plt.axhline(y=1, color='g', linestyle='-')
    plt.text(5,1,"Maximum Value")
    plt.axhline(y=-1, color='r', linestyle='-')
    plt.text(3,-1,"Minimum Value")
    plt.grid(True)
    plt.title("Interactive sinusoidal functions")
    plt.ylabel("sin"+"("+chr(952)+"+"+chr(966)+")")
    plt.xlabel(chr(952))
    plt.legend()
    plt.show()
```

```
s = Sines()
s.addSine(0)
s.addSine(90)
s.shiftRight(45)
#s.shiftLeft(45)
#s.show()
```



To shift left:

```
def shiftLeft(self,sl):     #To shift graph left by the amount of offset mentioned while calling the method.
    self.sl = sl * (np.pi/180)
    self.t = np.arange(0,2*np.pi,0.025*np.pi)
    plt.plot(self.t,np.sin(self.t+self.sl),label=chr(966)+"="+"0"+chr(176))
    #plt.plot(x,np.sin(x),marker='^',label="sin")
    if self.phi != 0:
        plt.plot(self.t,np.sin(self.t+self.deg+self.sl),label=chr(966)+"="+str(self.phi)+chr(176))
    #plt.plot(x,np.cos(x),marker='+',label="cos")
    #plt.plot(self.t,np.sin(self.t+self.deg),label=chr(966)+"="+"90"+chr(176))
    #plt.xticks(t)
    plt.axhline(y=1, color='g', linestyle='-')  #To mark the maximum level
    plt.text(5,1,"Maximum Value")
    plt.axhline(y=-1, color='r', linestyle='-') #To mark the minimum level
    plt.text(3,-1,"Minimum Value")
    plt.grid(True)
    plt.title("Interactive sinusoidal functions")
    plt.ylabel("sin"+"("+chr(952)+"+"+chr(966)+")")
    plt.xlabel(chr(952))
    plt.legend()
    plt.show()
```

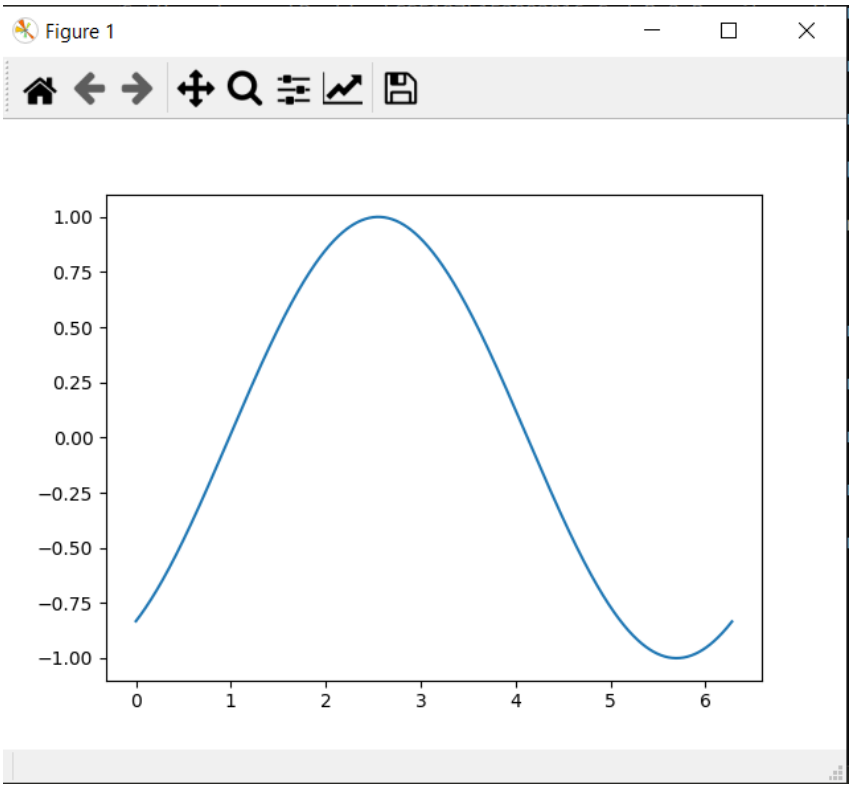


4) **Q5.py:** To animate sine curve by interacting with it.

```
#To interact with the animation of sine curve.
    def interact(self):
        fig, ax = plt.subplots()
        #NumPy is a Python library used for working with arrays.
        #arange() is an inbuilt numpy function that returns an ndarray object containing evenly spaced values within a defined interval.
        x = np.arange(0, 2*np.pi, 0.01)
        line, = ax.plot(x, np.sin(x))
    #To be able to modify the values of running and direction, I assigned them to anim.
    #For the manual update, I'm accessing anim's generator object that FuncAnimation uses to update the plot.
    #This ensures that when I resume the animation, it starts from the active frame rather than from where it was originally paused.
        def update_time():
            t = 0
            t_max = 10
            while t<t_max:
                t = t + anim.direction
                yield t

        def animate(i):
            line.set_ydata(np.sin(x - i/10.0))
            return line,
        #To simulate animation of sine curve when pressed certain keys as required in the question.
        def on_press(event):
            if event.key.isspace():
                if anim.running:
                    anim.event_source.stop()
                else:
                    anim.event_source.start()
                anim.running ^= True
            elif event.key == 'a':
                anim.direction = -1
            elif event.key == 'd':
                anim.direction = +1

            # Manually update the plot
            if event.key in ['a','d']:
                t = anim.frame_seq.next()
                animate(t)
                plt.draw()
```

```
        #To receive events, we need to write a callback function and then connect our function to the event manager, which is part of the FigureCanvasBase.
        #The FigureCanvas method mpl_connect() is used for event handling and picking.
        fig.canvas.mpl_connect('key_press_event', on_press)
        anim = ani.FuncAnimation(fig, animate, frames=update_time,
                                 interval=100, repeat=True)
        anim.running = True
        anim.direction = -1
        plt.show()

s = Sines()
s.interact()
```

A still from the output:



Thank you