As per the assignment I have created Four python files named **Q2.py, Q3.py Q4.py and Q5.py**.
This report contains detailed explanation of my code.

Description of Q2.py :

Q2.py file contains **PieChart** class.

Properties of Sines class :
• It contains 1 constructor and 3 methods.

__init__(self, dict) method or constructor of PieChart Class :

- **__init__()** method is the first method that will be invoked at the time of **object creation**
- For handling exceptions, t**ry except** block is used. A PieChart object can be created only if defined with Dictionary **dict key** is a **string** and the **key value** is of type integer or greater than 0 .Otherwise it will raise **Exception**.
- **except** block will catch the exception raised in **try** block and it will print the **type of exception and the message to user for that exception.**

__add__(self, other) method of PieChart Class :

- In this method we **overload operator** to **add** values to **dict** by passing **other** as parameter. We check if the value is of tuple type or dictionary type.
- If the type is of **tuple** then a **flag** is used to check if the key of the tuple is present in the PieChart object. If it is present, the value adds and if not present then it is being added as a new key value pair in the dictionary of PieChart object.
- If the type of other is Piechart that is **dictionary**, then **Counter** is used to add 2 dictionaries.
- The operator will return a PieChart object.

__sub__(self, other) method of PieChart Class :

- In this method we overload operator to **subtract** a dictionary by passing a key as other in string format.
- We use **pop** method to remove the key from the PieChart object according to the key passed as other.

Screenshots of Test Cases outputs:

PS E:\python\Python codes> python -u "e:\python\Python codes\Q2.py"

Testcase 1 :

Input :
p = PieChart({1, 23})

Output :
<class 'Exception'>
Label should be string

Testcase 2 :

Input :
p = PieChart({'Frog': '30'})

Output :
<class 'Exception'>
Value should be a positive numeric

Testcase 3:

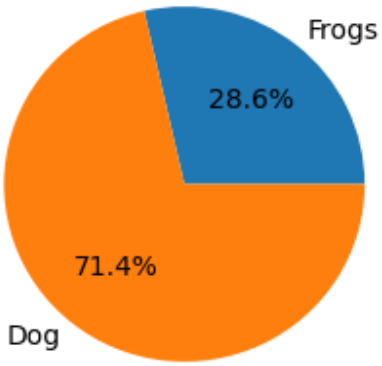
Input :
p = PieChart({'Frog': -10})

Output :
<class 'Exception'>
Value should be a positive numeric

Testcase 4 :

Input :
p = PieChart({'Frogs': 10, 'Dog': 25})
p.show()

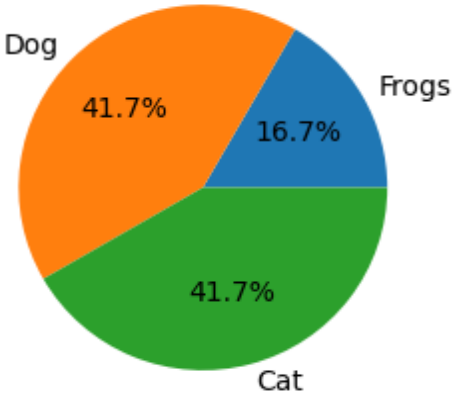
Output :



Testcase 5 :

Input :
p = PieChart({'Frogs': 10, 'Dog': 25})
p = p + ('Cat', 25)
p.show()

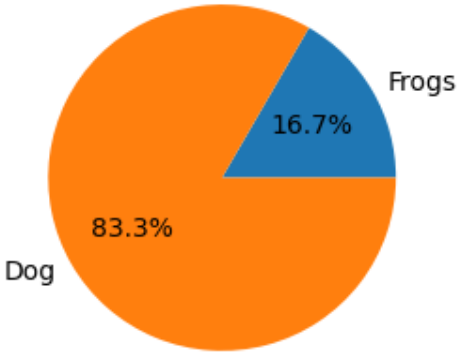
Output :



Testcase 6 :

Input :
p = PieChart({'Frogs': 10, 'Dog': 25})
p = p + ('Dog', 25)
p.show()

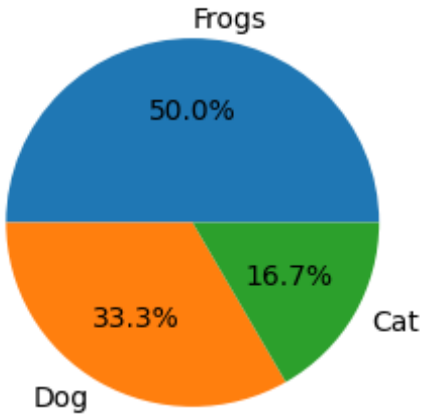
Output :



Testcase 7 :

Input :
p = PieChart({'Frogs': 10, 'Dog': 25})
p = p + PieChart({'Frogs': 20, 'Cat': 10})
p.show()

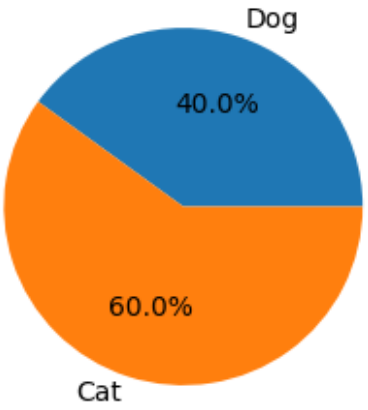
Output :



Testcase 8 :

Input :
p = PieChart({'Frogs': 10, 'Dog': 20, 'Cat':30})
p = p - 'Frogs'
p = p - 'Lions'
p.show()

Output :



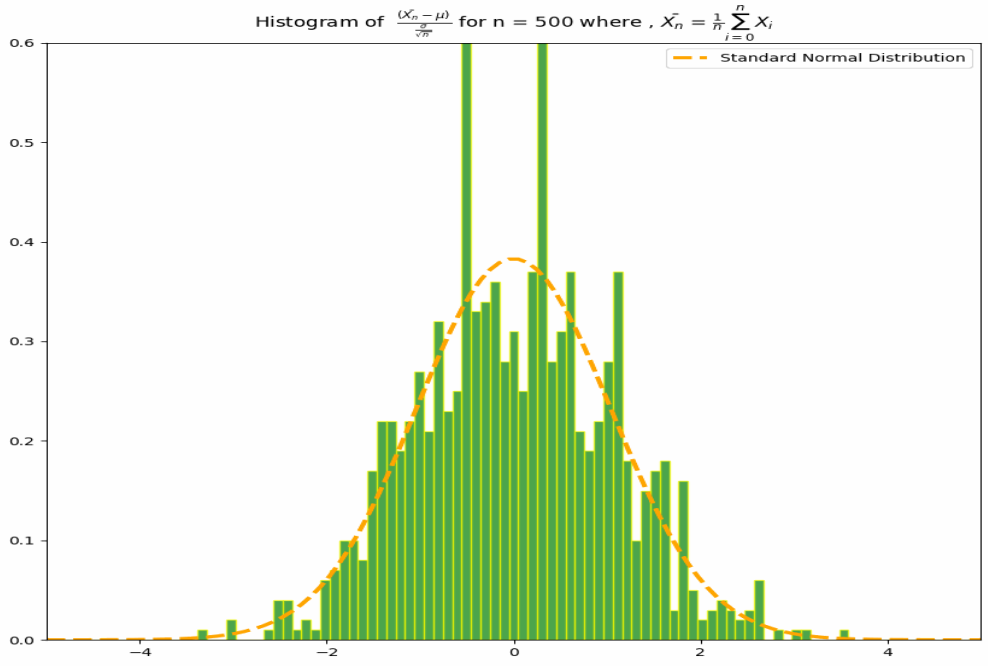
Description of Q3.py :

- The **Q3.py** file starts with two variables **n** and **k** where **n** is the number of random variables and **k** is the number of values to be taken for each random variable.
- Then I created a **Bernoulli distribution** with probability of success 0.5.
- Then the **n , k** are initialised to 500 and 1000 , where **n** is the number of random variables of **bernoulli's equation** in one sample and **K** is no of random samples respectively .
- Now the **figure** and **axis subplots** are initialized as **fig** and **ax**.
- **animate(frame)** function is created to send the frame numbers and accordingly the number of samples increases such that it becomes a very large value and plots the histogram such that it is comparable to a standard normal distribution.
- The **animate** method is used to create animation which is called by **FuncAnimation** method**.**
    - The animate method takes frame number as an argument .
    - In each call to **animate** method , **n** is incremented by 500.
    - The **mean** μ as **u** and **standard deviation** σ as **s** is calculated from this **data**.
    - The **sample_means** of each sample $\mathbf{X_i}$ is determined that is $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$.
    - The standard normal distribution that is, $\frac{\overline{X}_n - \mu}{\sigma/\sqrt{n}}$ is calculated and taken as **Y**.
    - The **maximum and minimum X limits** are taken as 10 times of Standard deviation in the figure to show all the data sets.
    - The **normal distribution curve** is plotted by taking the mean and standard deviation of the histograms.
    - The **histogram** is plotted with all the data Y.
    - The height of each bar of the histogram is scaled down by dividing them by 100.

- legend was added at the upper right corner of the plot for the normal curve .
- The top value of the plot is fixed .
- The **title** is set inside the **animate** method so that , when the **FuncAnimation()** method calls **animate** method, it continuously animates and plots the curves and every time the **title** changes.
- The **animate(frame)** returns **ax .**

- The plot is shown on the output screen.

Output:



For Gif histogram please check CLT.gif file.

Description of Q4.py :

Q4.py file contains **Sines** class.

Properties of Sines class :
• It contains 1 constructor and 4 methods.

__init__(self) constructor of Sines Class :

- **__init__()** method is the first method that will be invoked at the time of **object creation.** In the Sines class **__init__()** function is used to initialize 2 attributes of array type **ys**, **rads**.
- **ys[]** stores tuples of (**rad,y)** where rad is the updated radian value for a sine curve y
- **rads[]** stores radian values of each plot **y** when it was created.
- **x** is linspace .

addSine(self, deg) method of Sines Class :

- This method takes a **phase value** in degree as an argument **deg**.
- It converts phase angle value to radian and stores it in **rads**.
- Then it **creates a Sine curve** with phase angle and x-axis values and stores it in **ys** in a (rad,y) mananer.

shiftRight(self, deg) method of Sines Class :

- This method takes the degree value, by which all the curves would be **shifted to the right** side of the plot, as an argument **deg** and converts it to radian value.
- It modifies the previously stored radian values of all the curves and stores it in the same **rads**.
- It modifies the **ys** curves previously stored.

shiftLeft(self, deg) method of Sines Class :

- This method takes the degree value **deg** , by which all the curves would be **shifted to the left side** of the plot, as an argument **deg** and converts it to radian value.
- It modifies the previously stored radian values of all the curves and stores it in the same **rads**.
- It modifies the **ys** curves previously stored.

show(self) method of Sines Class :

- This method **creates the plots** in the figure by taking the curves stored and labels the curve by **legend()** method.
- This method is used to create the **figure**, **subplots**, **pre-defined axes** and their **limits**, **labels** of axes, **grids**, **and title** of the plot.
- This method highlights **3 straight lines** having maximum, minimum possible values of Sine curve and 0 line of Sine curve.
- Finally, it **displays the figure** after every modification.

Screenshots of Test Cases outputs:

Testcase 1 :

Input :

```
s = Sines()
s.addSine(0)
s.addSine(90)
```

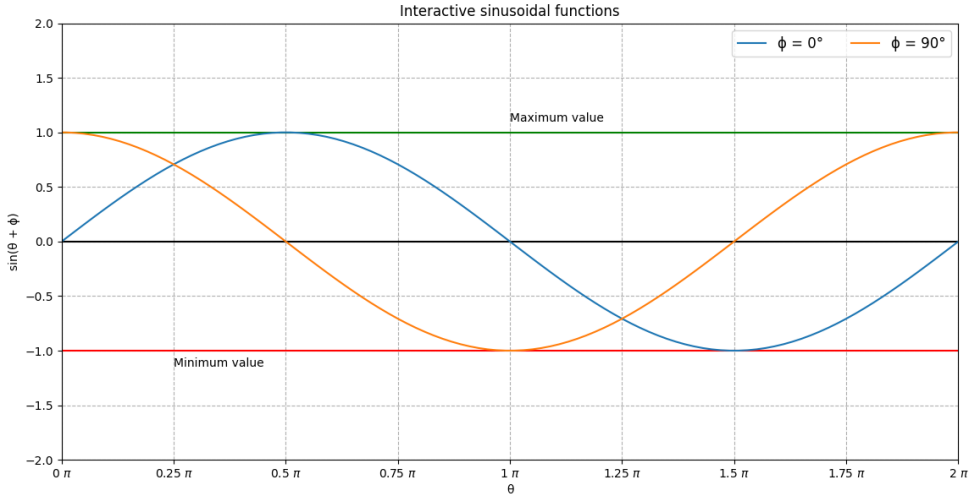
Output :

Testcase 2 :

Input :

```
s = Sines()
s.addSine(0)
s.addSine(90)
s.show()
```

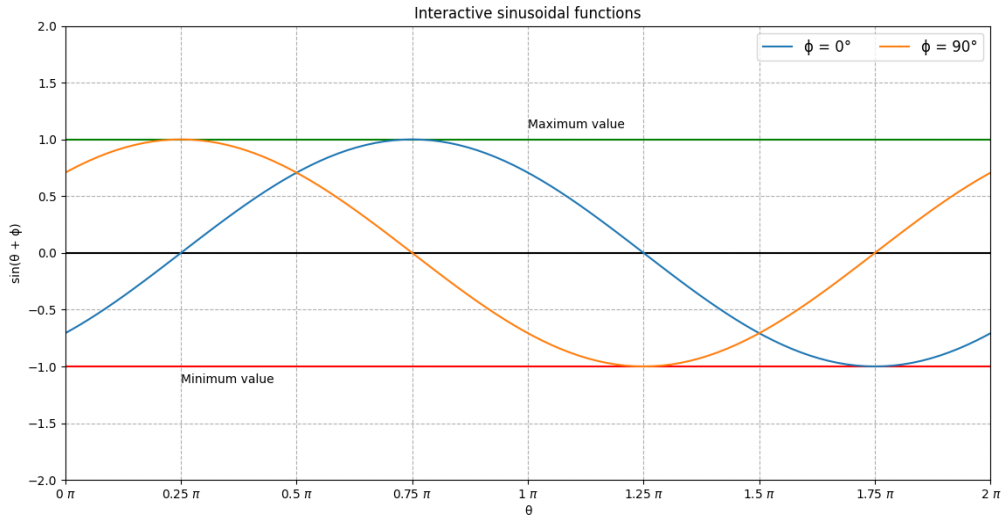
Output :



Testcase 3 :

Input :
s = Sines()
s.addSine(0)
s.addSine(90)
s.show()
s.shiftRight(45)

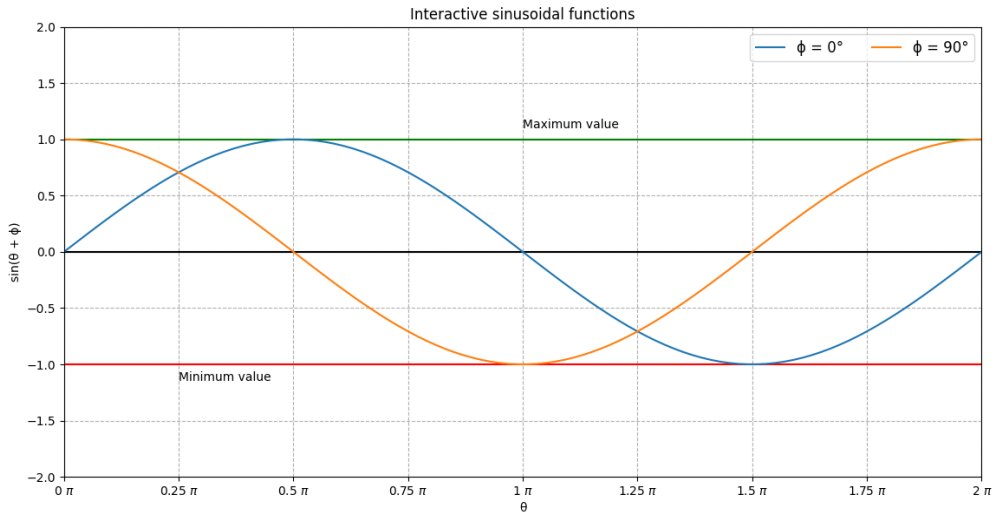
Output :



Testcase 4 :

Input :
s = Sines()
s.addSine(0)
s.addSine(90)
s.show()
s.shiftRight(45)
s.shiftLeft(45)

Output :



Description of Q5.py :

Q5.py file contains **Sines** class.

Properties of Sines class :

• It contains 1 constructor and 4 methods.

__init__(self) constructor of Sines Class :

- **__init__()** method is the first method that will be invoked at the time of **object creation.** In the Sines class **__init__()** function is used to initialize 2 attributes of array type **ys**, **rads**.
- **ys[]** stores tuples of (**rad,y)** where rad is the updated radian value for a sine curve y
- **rads[]** stores radian values of each plot **y** when it was created.
- **x** is linspace .
- This method is used to create the **figure**, **subplots**, **pre-defined axes** and their **limits**, **labels** of axes, **grids**, **and title** of the plot.
- This method highlights **3 straight lines** having maximum, minimum possible values of Sine curve and 0 line of Sine curve.
- It initializes **right , left , paused** variables to **False.**
- **paused** indicates the plot is paused or not , **right** and **left** indicates the plot is moving towards right or **left**.

addSine(self, deg) method of Sines Class :

- This method takes a **phase value** in degree as an argument **deg**.
- It converts phase angle value to radian and stores it in **rads**.
- Then it **creates a Sine curve** with phase angle and x-axis values and stores it in **ys** in a (rad,y) mananer.

show(self) method of Sines Class :

- This method **creates the plots** in the figure by iterating **ys [] ,**where the curves are stored and labels the curve by **legend()** method.
- Finally, it **displays the figure** after every modification.

interact(self) method of Sines Class:

- This method is responsible for interaction with the plot on key press events .
- Here I used **mpl_connect()** method for event handling and for every keypress event it will call **interaction(self, event)** method to handle specific use cases of event handlings mentioned in the question , like moving the plot to right or left , pause or resume it and resetting the plot.
- Here the **FuncAnimation()** method is used to create animation for the plot . It calls the **animate(self,frame)** method to create the animation .
- **show(self)** is called to show the plot .

interaction(self , event) method of Sines Class:

- The **interaction(self,event)** method is used to capture the Keypress events and take necessary actions .
- If **'spacebar'** is pressed , it will toggle between the **paused** and **resume** state of the plot by using **resume**() and **pause**() method.
- if **'a'** is pressed , it will **resume** the plot if it is in **paused** state, **set** the **right** value to **True, left** value to **False , paused** value to **False ,** these values will be used to determine in which direction the curve should move.
- if **'d'** is pressed , it will **resume** the plot if it is in a **paused** state, **set** the **right** value to **False, left** value to **True** and **paused** value to **False .**
- if **'r'** is pressed , it will **pause** the plot if it is in the **resumed** state, **set** the **right** value to **False, left** value to **False** and **paused** value to **True.**

animate(self , frame) method of Sines Class:

- **animate(self, frame)** method is called by the **FuncAnimation()** method to create animation. It takes the frame number as the **frame** parameter .
- **v** variable is used to to set the x-limit , xticks to move the plot horizontally.
- If **left** is **True ,** decrement **v** by **0.01.**
- If **right** is **True ,** increment **v** by **0.01.**
- If **left** is **false** and **right** is **false ,** set **v** to **0 .**
- set x-limit , x-ticks , position of max_text , min_text using **v** so that for each call to animate function , the plot changes accordingly **.**
- **animate(self, frame)** returns **v.**

Output:

Please run the Q5.py file .