EX:9 Build a Scientific Calculator in tkinter

AIM:

To build a simple scientific calculator by the use of python library tkinter .

Requirements:

1. Jupyter Notebook

Coding:

Importing library

```
from tkinter import *
import math
```

Creating root window

```
root = Tk()
root.title("ScientificCalculator")
root.configure(background=
 'black')
 root.geometry("491x724")
calc = Frame(root)
calc.grid()
```

Creating classes for

```
 buttons class Calc():
def __init__(self):
self.total=0
self.current=''
self.input_value=True
      self.check_sum=False
self.op=''
      self.result=False

   def numberEnter(self,
num):       self.result=False
firstnum=txtDisplay.get()
secondnum=str(num)
if self.input_value:
         self.current = secondnum
self.input_value=False
else:
 if secondnum == '.':
if secondnum in firstnum:
               return
         self.current = firstnum+secondnum
      self.display(self.current)
```

```
   def sum_of_total(self):
self.result=True
self.current=float(self.current)
if self.check_sum==True:
self.valid_function()        else:
self.total=float(txtDisplay.get())

   def display(self, value):
txtDisplay.delete(0, END)
txtDisplay.insert(0, value)

   def display1(self, value):
txtDisplay1.delete(0, END)
       txtDisplay1.insert(0, value)



   def valid_function(self):
if self.op == "add":
self.total += self.current
if self.op == "sub":
self.total -= self.current
if self.op == "multi":
self.total *= self.current
if self.op == "divide":
self.total /= self.current
if self.op == "mod":
```

```
self.total %= self.current
self.input_value=True
self.check_sum=False
        self.display1(self.total)

    def operation(self, op):
        self.current =
float(self.current)
if self.check_sum:
self.valid_function()
 elif not self.result:
self.total=self.current
self.input_value=True
self.check_sum=True
self.op=op
        self.result=False

    def Clear_Entry(self):
self.result = False
self.current = "0"
self.display(0)
self.display1(0)
        self.input_value=True

    def pi(self):
        self.result =  False
```

```
        self.current = math.pi
        self.display1(self.current)

    def cos(self):
self.result = False
        self.current = math.cos(math.radians(float(txtDisplay.get())))
self.display1(self.current)

    def sin(self):
self.result = False
        self.current = math.sin(math.radians(float(txtDisplay.get())))
self.display1(self.current)

    def tan(self):
self.result = False
        self.current =
math.tan(math.radians(float(txtDisplay.get())))
self.display1(self.current)     def squared(self):
self.result = False
        self.current = math.sqrt(float(txtDisplay.get()))
self.display1(self.current)

added_value = Calc()
```

Creating Display for the calculator

```
txtDisplay = Entry(calc, font=('Helvetica',20,'bold'),bg='black',fg='white',
bd=20,                  width=30,justify=RIGHT)
txtDisplay.grid(row=0,column=0, columnspan=4, pady=1)
txtDisplay.insert(0,"0")
txtDisplay1 = Entry(calc, font=('Helvetica',20,'bold'),bg='black',fg='white',
bd=20,  width=30,justify=RIGHT)
txtDisplay1.grid(row=1,column=0, columnspan=4, pady=1)
txtDisplay1.insert(0,"0")
```

Creating numberpad

```
numberpad = "123456789" i=0 btn = [] for j in range(3,6):
for k in range(3):        btn.append(Button(calc, width=6,
height=2, bg='violet',fg='black', font=('Cambria',20,'bold'),
                    bd=4,text=numberpad[i]))
btn[i].grid(row=j, column= k, pady = 1)
      btn[i]["command"]=lambda
x=numberpad[i]:added_value.numberEnter(x)
 i+=1
```

Creating buttons

```
btnClear = Button(calc, text=chr(67),width=6,
height=2,bg='lightblue',fg='black', font=('Helvetica',20,'bold')
              ,bd=4, command=added_value.Clear_Entry).grid(row=2, column= 3,
pady = 1)
```

```
btnAdd = Button(calc, text="+",width=6,
height=2,bg='orange',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=lambda:added_value.operation("add")
).grid(row=6, column= 3, pady = 1)

btnSub = Button(calc, text="-",width=6,
height=2,bg='orange',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=lambda:added_value.operation("sub")
).grid(row=5, column= 3, pady = 1)

btnMul = Button(calc, text="x",width=6,
height=2,bg='orange',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=lambda:added_value.operation("multi")
).grid(row=3, column= 3, pady = 1)

btnDiv = Button(calc, text="/",width=6,
height=2,bg='orange',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=lambda:added_value.operation("divide")
).grid(row=4, column= 3, pady = 1)

btnZero = Button(calc, text="0",width=6,
height=2,bg='violet',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=lambda:added_value.numberEnter(0)
).grid(row=6, column= 0, pady = 1)
```

```
btnDot = Button(calc, text=".",width=6,
height=2,bg='violet',fg='black', font=('Helvetica',20,'bold'),
            bd=4,command=lambda:added_value.numberEnter(".")
           ).grid(row=6, column= 1, pady = 1)
btnOpenbrace = Button(calc, text="(",width=6,
height=2,bg='green',fg='black', font=('Helvetica',20,'bold'),
            bd=4,command=lambda:added_value.numberEnter("(")
).grid(row=7, column= 1, pady = 1)

btnClosbrace = Button(calc, text=")",width=6,
height=2,bg='green',fg='black', font=('Helvetica',20,'bold'),
            bd=4,command=lambda:added_value.numberEnter(")")
).grid(row=7, column=2, pady = 1)

btnEquals = Button(calc, text="=",width=6,
height=2,bg='red',fg='black', font=('Helvetica',20,'bold'),
            bd=4,command=added_value.sum_of_total).grid(row=7, column=3 ,
pady = 1)

btnsq = Button(calc, text="\u221A",width=6,
height=2,bg='violet',fg='black', font=('Helvetica', 20,'bold'),
           bd=4,command=added_value.squared).grid(row=6, column=2, pady =
1)

btnPi = Button(calc, text="Pi",width=6,
height=2,bg='green',fg='black', font=('Helvetica',20,'bold'),
```

```
        bd=4,command=added_value.pi).grid(row=7, column= 0,
pady = 1)

btnCos = Button(calc, text="cos",width=6,
height=2,bg='yellow',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=added_value.cos).grid(row=2, column= 1,
pady = 1)

btnsin = Button(calc, text="sin",width=6,
height=2,bg='yellow',fg='black', font=('Helvetica',20,'bold'),
        bd=4,command=added_value.sin).grid(row=2, column= 0,
pady = 1)

btntan = Button(calc, text="tan",width=6,
height=2,bg='yellow',fg='black',font=('Helvetica',20,'bold'),
       bd=4,command=added_value.tan ).grid(row=2, column=2 ,
pady = 1)

Call root window
root.mainloop()
```

Result: Thus the way we declare and execute the Scientific Calculator in tkinter using Python is verified successfully

EX:10 Fundamentals of Pygame and Build a simple snake game in Python

Aim:
To Build a simple snake game by the use of python library pygame

Requirements:

1. Jupyter Notebook

Coding:

Fundamentals of Pygame

```
import pygame
pygame.init()
white = (0, 255, 255)
display_surface = pygame.display.set_mode((920,600))
pygame.display.set_caption('HI AI Students')

image =
pygame.image.load('C:\Users\aedpu\OneDrive\Desktop\DFBEagle.png'
)

while True:
 display_surface.fill(white)
 display_surface.blit(image, (0, 0))
 for event in pygame.event.get():
 if event.type == pygame.QUIT:
 pygame.quit()
pygame.display.update()
```

Build a Classical snake game in Python:

1. Importing Library

```
import pygame
import time
import random
```

2. Setting Snake Speed

```
snake_speed = 15
```

3. Setting Window Size

```
window_x = 720
window_y = 480
```

4. Defining Colors

```
black = pygame.Color(0, 0, 0)
white = pygame.Color(255, 255, 255)
red = pygame.Color(255, 0, 0)
green = pygame.Color(0, 255, 0)
blue = pygame.Color(0, 0, 255)
```

5. Initialising Pygames

```
pygame.init()
```

6. Initialising Game Window

```
pygame.display.set_caption('Snakes')
game_window = pygame.display.set_mode((window_x, window_y))
```

7. Setting FPS Controller

```
fps = pygame.time.Clock()
```

8. Defining Snake Default Position

```
snake_position = [100, 50]
```

9. Defining 1st 4 snake body blocks

```
snake_body = [[100, 50],
              [90, 50],
              [80, 50],
              [70, 50]]
```

10. Fruit Position

```
fruit_position = [random.randrange(1, (window_x//10)) * 10,
```

```
 random.randrange(1, (window_y//10)) * 10]
fruit_spawn = True
```

11. Setting Default Snake Direction

Right Direction

```
direction = 'RIGHT'
change_to = direction
```

12. Set Initial Score

```
score = 0
```

13. Function - Screen Display

```
def show_score(choice, color, font, size):
score_font = pygame.font.SysFont(font, size)
 score_surface = score_font.render('Score : ' + str(score), True, color)
score_rect = score_surface.get_rect()
game_window.blit(score_surface, score_rect)
```

14. Function - Game Over

```
def game_over():

my_font = pygame.font.SysFont('times new roman', 50)
game_over_surface = my_font.render(
 'Your Score is : ' + str(score), True, red)
game_over_rect = game_over_surface.get_rect()
game_over_rect.midtop = (window_x/2, window_y/4)
game_window.blit(game_over_surface, game_over_rect)

pygame.display.flip()
 time.sleep(2)

pygame.quit()
quit()
```

15. Function - Main Game Function

```
# Main Function
while True:

 # handling key events
 for event in pygame.event.get():
     if event.type == pygame.KEYDOWN:
       if event.key == pygame.K_UP:
          change_to = 'UP'
       if event.key == pygame.K_DOWN:
          change_to = 'DOWN'
       if event.key == pygame.K_LEFT:
          change_to = 'LEFT'
       if event.key == pygame.K_RIGHT:
         change_to = 'RIGHT'

if change_to == 'UP' and direction != 'DOWN':
   direction = 'UP'
if change_to == 'DOWN' and direction != 'UP':
  direction = 'DOWN'
if change_to == 'LEFT' and direction != 'RIGHT':
  direction = 'LEFT'
if change_to == 'RIGHT' and direction != 'LEFT':
  direction = 'RIGHT'

# Moving the snake
if direction == 'UP':
   snake_position[1] -= 10
if direction == 'DOWN':
   snake_position[1] += 10
if direction == 'LEFT':
   snake_position[0] -= 10
if direction == 'RIGHT':
  snake_position[0] += 10
```

```
snake_body.insert(0, list(snake_position))
 if snake_position[0] == fruit_position[0] and snake_position[1] ==
fruit_position[1]:
 score += 10
 fruit_spawn = False
 else:
 snake_body.pop()

 if not fruit_spawn:
 fruit_position = [random.randrange(1, (window_x//10)) * 10,
 random.randrange(1, (window_y//10)) * 10]

 fruit_spawn = True
 game_window.fill(black)

 for pos in snake_body:
 pygame.draw.rect(game_window, green,
 pygame.Rect(pos[0], pos[1], 10, 10))
 pygame.draw.rect(game_window, white, pygame.Rect(
 fruit_position[0], fruit_position[1], 10, 10))

 # Game Over conditions
 if snake_position[0] < 0 or snake_position[0] > window_x-10:
 game_over()
 if snake_position[1] < 0 or snake_position[1] > window_y-10:
 game_over()

 # Touching the snake body
 for block in snake_body[1:]:
 if snake_position[0] == block[0] and snake_position[1] == block[1]:
 game_over()

 # displaying score countinuously
 show_score(1, white, 'times new roman', 20)
```

```
# Refresh game screen
pygame.display.update()

# Frame Per Second /Refres Rate
fps.tick(snake_speed)
```

Result:

Thus the way we declare and execute the simple snake game in Python is verified successfully

EX:11 Building Simple Flask app

Aim:
To Build the simple Flask App by the use of python library FLASK

Requirements:

1. Jupyter Notebook

Coding:

Hello World Web Page:

```
# Importing required library
from flask import Flask

# Defining flask name
hwapp = Flask(__name__)

# Creating main page
@hwapp.route('/')
def index():
 return 'Hello World'
   if __name__ == "__main__":
 hwapp.run()
```

Greeting Web Development:

```
from flask import Flask, render_template, request, flash

app = Flask(__name__)
app.secret_key = "key_set123"

@app.route("/")
def index():
   flash("What's your name?")
```

```
    return render_template("index.html")

@app.route("/greet", methods=['POST', 'GET'])
def greeter():
    flash("Hi " + str(request.form["name-input"]) + ", nice to see you!")
    return render_template("index.html")

@app.route("/contact", methods=['POST', 'GET'])
def contact():
    flash("Mobile: +91 xxxxxxxxxx")
    flash("Email: xyz@abc.com")
    return render_template("index copy.html")

if __name__ == '__main__':
    app.run
```

Result:
Thus the way we declare and execute the simple Flask App in Python is verified successfully

EX:12 Build a student Digital profile using FLASK

Aim:

To Build a student Digital profile by the use of python library FLASK

Requirements:

1. Jupyter Notebook

Coding:

```
from flask import Flask, render_template, flash, redirect, url_for, session


app=Flask(__name__)
app.config['SECRET_KEY']='some_random_secret'
@app.route("/") def
index():
   return render_template("index1 DB copy.html")
@app.route("/scholarship") def
scholarship():
   return render_template("form.html")
@app.route("/success", methods=['POST', 'GET'])
def ssuccess():
   return render_template("Success.html")
if __name__ == '__main__':
   app.run()
```

Result:

Thus the way we declare and execute the Student Digital profile using FLASK in Python is verified successfully