Garden program: GUI with three buttons, and mouse events, and objects of a class

```
from tkinter import *

#-------------------------------------------------
#Define the class of flowers
#A flower object contains
#    its position (x and y)
#    its height, and
#    whether it is blooming or not

class Flower :

    # Constructor: create a new flower at the specified position
    #  The constructor is always called __init__
    def __init__(self, xpos, ypos) :   # first parameter is the object being created
        self.x = xpos
        self.y = ypos
        self.height = 20;
        self.blooming = False

    # draw a flower
    def draw(self) :  # first parameter is the object that draw was called on
        stem = 3      # half the width of the stem
        top = self.y-self.height
        canvas.create_rectangle(self.x-stem, top, self.x+stem, self.y,
                                fill="green", outline="green")
            # rectangles specified by left, top, right, bottom .
        if self.blooming :
            radius = 15    # radius and colour of the flower head
            colour = "red"
        else :
            radius = 9     # radius and colour of the bud
            colour = "green"
        canvas.create_oval(self.x-radius, top-radius, self.x+radius, top+radius,
                           fill=colour, outline=colour)
            # ovals also specified by left, top, right, bottom .

    # make a flower higher
    def grow(self, increment) :
        self.height += increment

    # make a flower bloom
    def bloom(self) :
        self.blooming = True;

#-------------------------------------------------
# main program
#  Constructs the unser interface and contains the list of all the flower objects

garden = []  # The list of flower objects in the garden

# Function to set up the window and canvas to draw on
def setup() :
    global canvas       # the canvas will be accessed by lots of methods. Since some of those methods are called
                        # from buttons, it requires special mechanisms to pass it to all the methods that need to use it.
                        # Much cleaner to make it global
    window = Tk()       # make a new window
    canvas = Canvas(window, width=600, height=450, bg='white')   # make a new canvas
```

```
    canvas.pack()        #pack it into the window (won't show if you don't pack it)
    butGrow = Button(window, text="Grow", command=growAll)     # make new button for window
    butGrow.pack()    # pack it into the window
    butBloom = Button(window, text="Bloom", command=bloomAll)# make 2nd button for window
    butBloom.pack()
    Button(window, text="Clear", command=clear).pack() # make and pack button in one line

    canvas.bind("<ButtonRelease-1>", plant)    # make canvas respond to left mouse button
    window.mainloop()


# plant a new flower at the mouse click
def plant(event) :                      # parameter is a description of the mouse event  that invoked this
    flower = Flower(event.x, event.y)  # make the flower object
    garden.append(flower)               # put the flower object in the garden
    drawGarden()                        # redraw the garden

# make each flower in the garden grow
def growAll() :
    for flower in garden :
        flower.grow(10)
    drawGarden()

# Make each flower in the garden bloom
def bloomAll() :
    for flower in garden :
        flower.bloom()
    drawGarden()

# Clear all the flowers from the garden and clear the canvas
def clear() :
    global garden
    garden = []
    canvas.delete(ALL)   # delete everything currently on the canvas
    canvas.update()      # necessary to make the changes in the canvas visible

#  Draw each flower in the garden
def drawGarden() :
    canvas.delete(ALL)
    for flower in garden :
        flower.draw()
    canvas.update()

#  Start the program by calling setup
setup()
```

Bouncing Ball program: GUI with one button, objects of a simple class, and animation

```
from tkinter import *
from random import random
import time

# Good programming practice is to make these *variables* rather than "magic numbers"
thrown into the code.
canvas_height = 600
canvas_width = 800
world_top = 20                          # 20 pixels from the top of the canvas
world_ground = canvas_height - 20       # 20 pixels from bottom of the canvas
world_left = 20                         # 20 pixels from the left of the canvas
world_right = canvas_width


# This class represents the bouncing ball class
class BouncingBall:

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.vx = random()+ .1
        self.vy = random()
        self.radius = (random() *10) + 4
        self.color = "black"

    #Get the x position of the ball
    def getX(self):
        return self.x

    def move(self):
        #Set the new x and y
        self.x += self.vx
        self.y += self.vy
        #Is y now below the ground?
        if self.y + self.radius >= world_ground:
            self.y = world_ground - self.radius#Sit it on ground level
            self.vy = self.vy * -1 #Reverse the y velocity
        #Is y above the top of the world?
        if self.y - self.radius <= world_top:
            self.y = world_top + self.radius #Make it so it is touching the top
            self.vy = self.vy * -1 #Reverse the y velocity


    # Draw the ball in the given canvas
    def draw(self):
        left = self.x - self.radius
        right= self.x + self.radius
        top = self.y - self.radius
        bot = self.y + self.radius
        canvas.create_oval(left, top, right, bot, fill=self.color)
```

```
#Code that controls the world that the bouncing balls are in
#The world (but maybe not GUI) could be a class as well, although at this stage it isn't.

bouncing_balls = []


def initialise_gui():
    global canvas, number_of_balls_field
    window = Tk()
    Label(window, text="How many balls?").pack()
    number_of_balls_field = Entry(window)
    number_of_balls_field.pack()
    Button(window, text="Reset", command=start_animation).pack()
    canvas = Canvas(window, height=canvas_height, width=canvas_width, bg="white")
    canvas.pack()
    draw_world_outline()
    window.mainloop()


#Makes new random bounching balls objects and puts them into the list
def list_of_balls(number_of_balls):
    balls = [] #Reset the bouncing balls list
    for i in range(number_of_balls):
        height = world_top+(world_ground-world_top)*random()
        ball = BouncingBall(world_left, height)
        balls.append(ball)
    return balls

def start_animation():
    #Read how many balls are needed
    count = int(number_of_balls_field.get())
    balls = list_of_balls(count)
    while balls: # While there are still balls in the list (balls are removed when they go over the edge of the screen)
        #time.sleep-(1)
        remaining = []
        for ball in balls:
            ball.move()
            if ball.getX() < world_right:
                remaining.append(ball)
        balls = remaining
        redraw_world(balls)

def redraw_world(balls):
    canvas.delete(ALL)
    draw_world_outline()
    for ball in balls:
        ball.draw()
    canvas.update()

def draw_world_outline():
    canvas.create_line(world_right, world_top, world_left, world_top, world_left,
                       world_ground, world_right, world_ground) #Line along the bottom

# This is the "main" method. I think this is all we need to do, as the rest of the program is event driven input
initialise_gui()
```

Banquet Table program: GUI with a label, one button, objects of a simple class, and mouse clicks that select objects

```
from tkinter import *

#  lets the user place and rearrange tables

# Represents a table with four chairs around it.
class Table() :
    def __init__(self, x, y) :
        self.xPos = x
        self.yPos = y
        self.rad = 30
        self.chairRad = 12

    def posOn(self, x, y) :
        return (self.xPos-self.rad <= x and x <= self.xPos+self.rad and
                self.yPos-self.rad <= y and y <= self.yPos+self.rad)

    def setPos(self, x, y) :
        self.xPos = x
        self.yPos = y

    def draw(self, canvas) :
        left = self.xPos - self.rad
        right= self.xPos + self.rad
        top  = self.yPos - self.rad
        bot  = self.yPos + self.rad
        self.drawChair(canvas, left, self.yPos)
        self.drawChair(canvas, right, self.yPos)
        self.drawChair(canvas, self.xPos, top)
        self.drawChair(canvas, self.xPos, bot)
        canvas.create_rectangle(left, top, right, bot, fill="brown")

    def drawChair(self, canvas, x, y) :
        left = x - self.chairRad
        right= x + self.chairRad
        top  = y - self.chairRad
        bot  = y + self.chairRad
        canvas.create_oval(left, top, right, bot, fill="black")




# set up the window
def main() :
    global canvas
    global tables
    window= Tk()
    Label(window, text="Click to move or place tables").pack()
    Button(window, text="Restart", command = restart).pack()
    canvas = Canvas(window, width=500, height=450, bg = 'white')
    canvas.bind("<Button-1>", mousepress)
    canvas.bind("<ButtonRelease-1>", mouserelease)
    canvas.pack()
    tables = []
    window.mainloop()
```

```
def restart() :
    global tables
    tables = []
    canvas.delete(ALL)
    canvas.update()

def mousepress(event):
    global selected
    for table in tables :
        if table.posOn(event.x, event.y) :
            selected = table
            return
    selected = None

def mouserelease(event):
    global selected
    if selected is None :
        tables.append(Table(event.x, event.y))
    else :
        selected.setPos(event.x, event.y)
    redrawTables()

def redrawTables() :
    canvas.delete(ALL)
    for table in tables :
        table.draw(canvas)
    canvas.update()


main()
```