Bouncing Ball Game Using Python

## Overview:

In this step-by-step tutorial, we will create a bouncing game where the ball bounces around the screen and the player will use a paddle to bounce off the ball. If the ball touches the bottom of the screen, the game ends.

For developing the game, we will use the Canvas class available in the Tkinter module. The Canvas is a rectangular area for drawing pictures or placing graphics, text or widgets for building the graphics in our program. We will add code for the different tasks:

* Create the game canvas
* Create the ball
* Animate the ball
* Create the paddle
* Move the paddle
* Detect collision

## Game Resources

### Setup Python Development Environment

Navigate to <https://www.python.org/downloads/> and download the relevant Python interpreter for your machine. This is the program that reads your Python code and carries out the instructions. If you are using a program that is not the same as above, please make sure that the interpreter includes the Tkinter module.

**Note**: Tkinter comes pre-installed with Python3.

## Game Design Concepts

In this section, we will review the different game design concepts that we will be using to build this game in Python.

### Understand the Computer Coordinate System

In mathematics, we use the Cartesian coordinate system to determine each point in the plane by identifying its x coordinate and y coordinate. The graph paper is placed at (0, 0) in the center. All numbers pointing up from y=0 are positive, and all numbers pointing down from y=0 are negative . If one had to represent (1, 1), it would be placed in the first quadrant (refer Figure 1).



Figure 1

However, the computer coordinate system behaves differently along the y axis. The x axis points from 0 to canvas width, and y axis points from 0 to canvas height. Unlike the Cartesian coordinate system, all numbers pointing down from y=0 are positive. If one had to represent (1, 1) using the computer coordinate system, it would be placed in the fourth quadrant (refer Figure 2). This explains why the Canvas widget has its upper left corner set at (0, 0).



Figure 2

#### Drawing Shapes on the Canvas

In this section, we will discuss the different create\_xx() methods that we will use for drawing different shapes for our game.

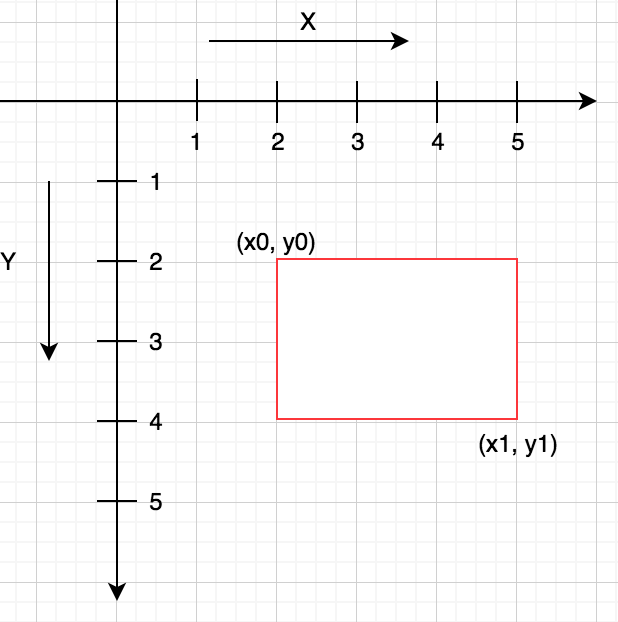
1. create\_rectangle(x0,y0,x1,y1): This method creates a rectangle at coordinates specified for the diagonal points of the rectangle, the top left corner (x0, y0) and the bottom right corner (x1, y1) (refer Figure 3). 

Figure 3

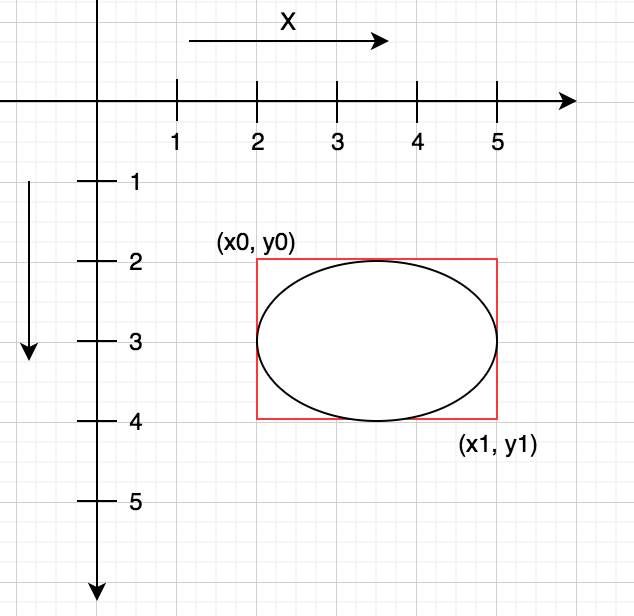
1. create\_oval(x0, y0, x1, y1): This method creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates, the top left corner (x0, y0) and the bottom right corner (x1, y1) of the bounding rectangle for the oval (refer Figure 4).

Figure 4

#### Moving Canvas Objects

The move(canvas\_object, x, y) method will be used for moving different objects on the Canvas screen. This method moves a Canvas object to a defined (x,y) location on the Canvas. A Canvas object is any valid image, drawing or text created with the help of the Canvas class.

### Make the Ball Bounce

For making the ball bounce, we are going to reverse the ball movement when it touches the screen boundary. For instance, if the ball touches the bottom of the screen, it must start going up. Similarly, if the ball touches the left side of the screen, it must reverse its direction and start going towards the right side of the screen.

#### Ball bounce: up and down

Managing the y coordinates of the ball (refer Figure 5), we are going to tell the interpreter that:

* If the ball hits the top of the screen (y0 <= 0), then stop moving the ball up. So, we will reset the self.y = 1.
* Similarly, if the ball hits the bottom of the screen (y1 >= height of the canvas), then stop moving it down. So we will reset the self.y = -1

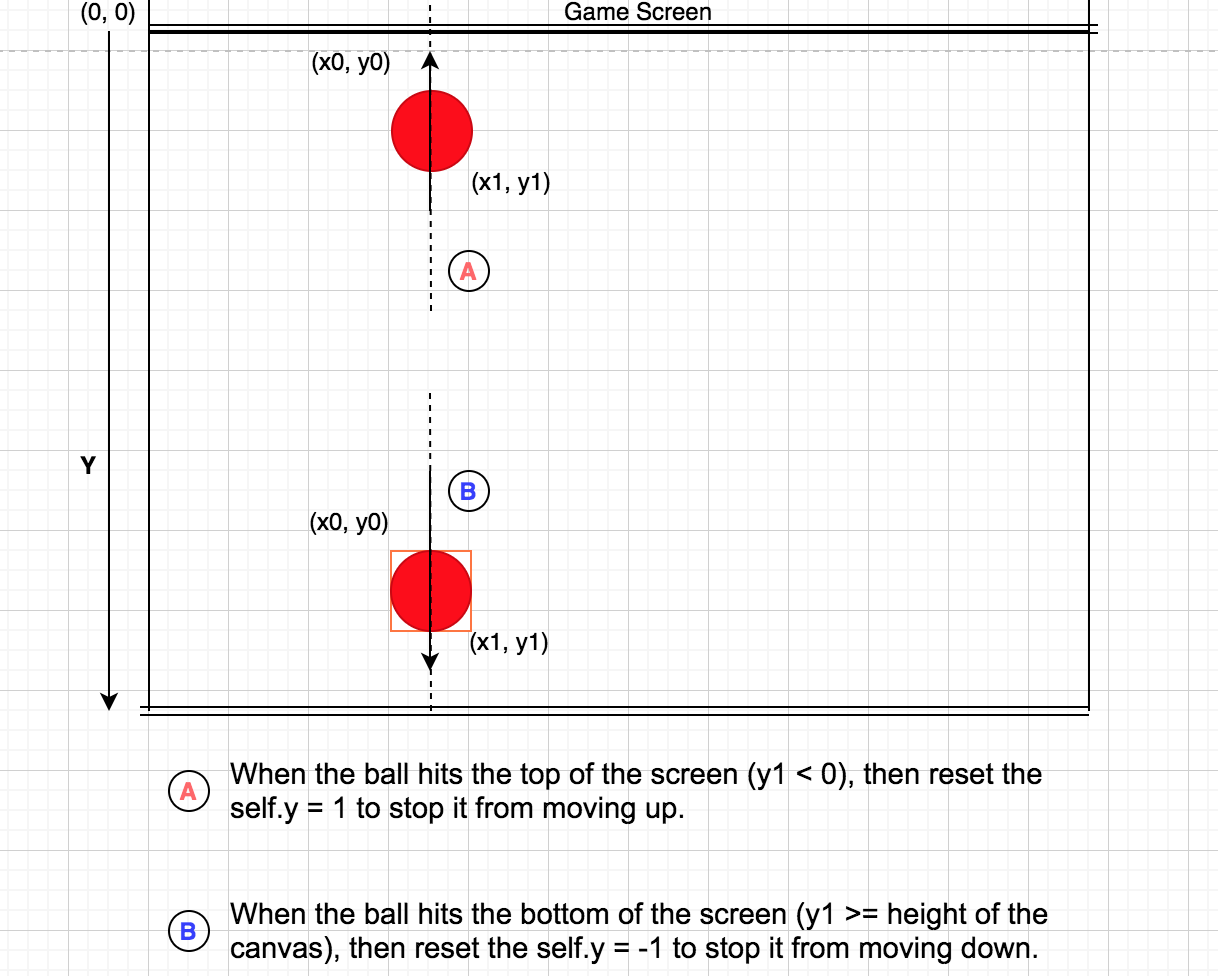


Figure 5

#### Ball bounce: left and right

Instead of the y coordinates, we will change the x coordinates to make the ball bounce left and right.

Managing the x coordinates of the ball (refer Figure 6), we are going to tell the interpreter that:

* If the ball hits the left of the screen (x0 <= 0), then stop moving the ball up. So, we will reset the self.x = 1.
* Similarly, if the ball hits the right of the screen (x1 >= width of the canvas), then stop moving it down. So, we will reset the self.x = -1.

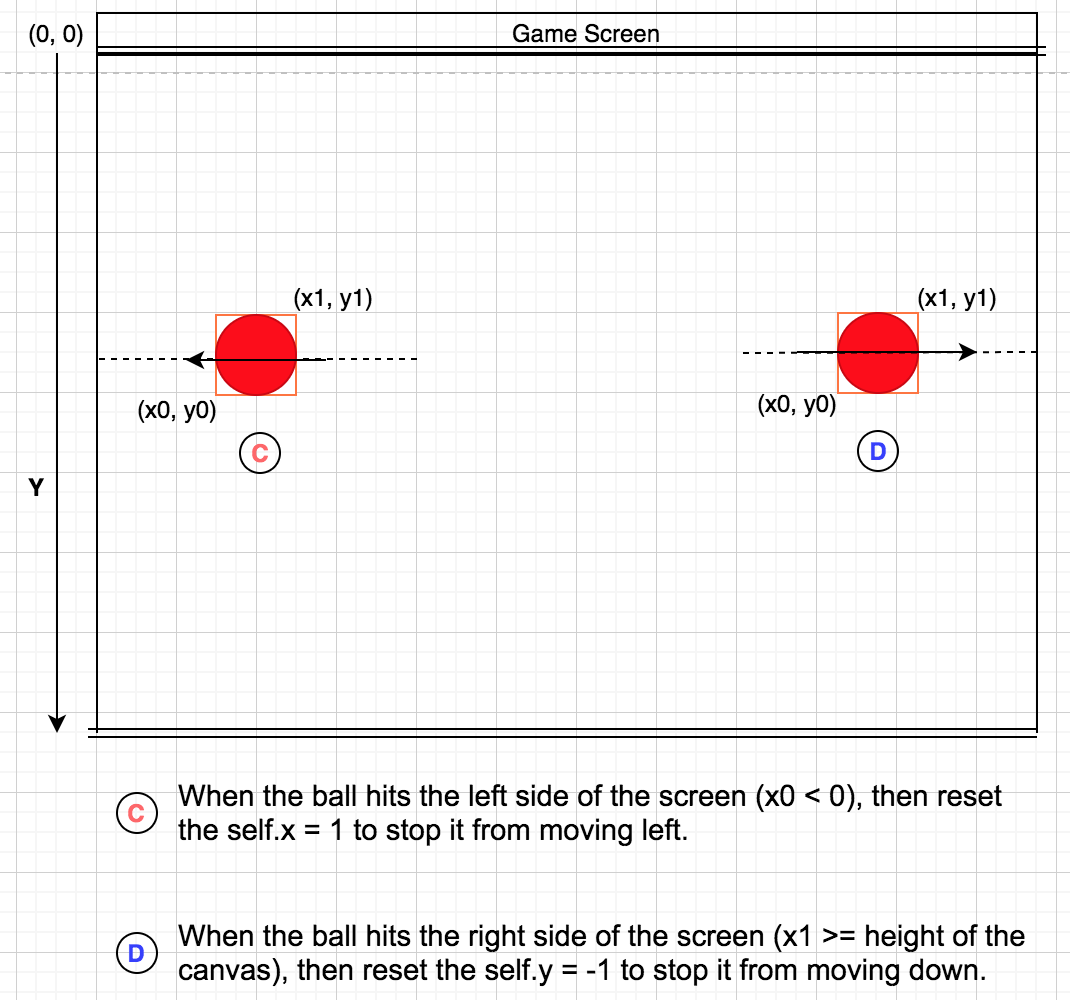


Figure 6

### Detect Ball Hitting the Paddle

The ball can hit the paddle while approaching from either the left side of the screen or right side of the screen. Each time the ball moves through the screen bouncing off the boundaries, its speed changes from 2 pixels to 10 pixels. If we just checked for the ball when it hits the top of the paddle, we may have leaped past the position and may not detect the hit.

For detecting the ball hitting the paddle, let us analyse the position of the ball and the paddle at the time of hit. In our game, ball position coordinates are (x0, y0, x1, y1) and paddle position coordinates are (a0, b0, a1, b1). The x axis points from 0 to canvas width, and y axis points from 0 to canvas height. Refer the respective diagrams for visual analysis for different conditions:

1. When the top of the ball hits the top of the paddle from the left side of the screen. Refer Figure 7

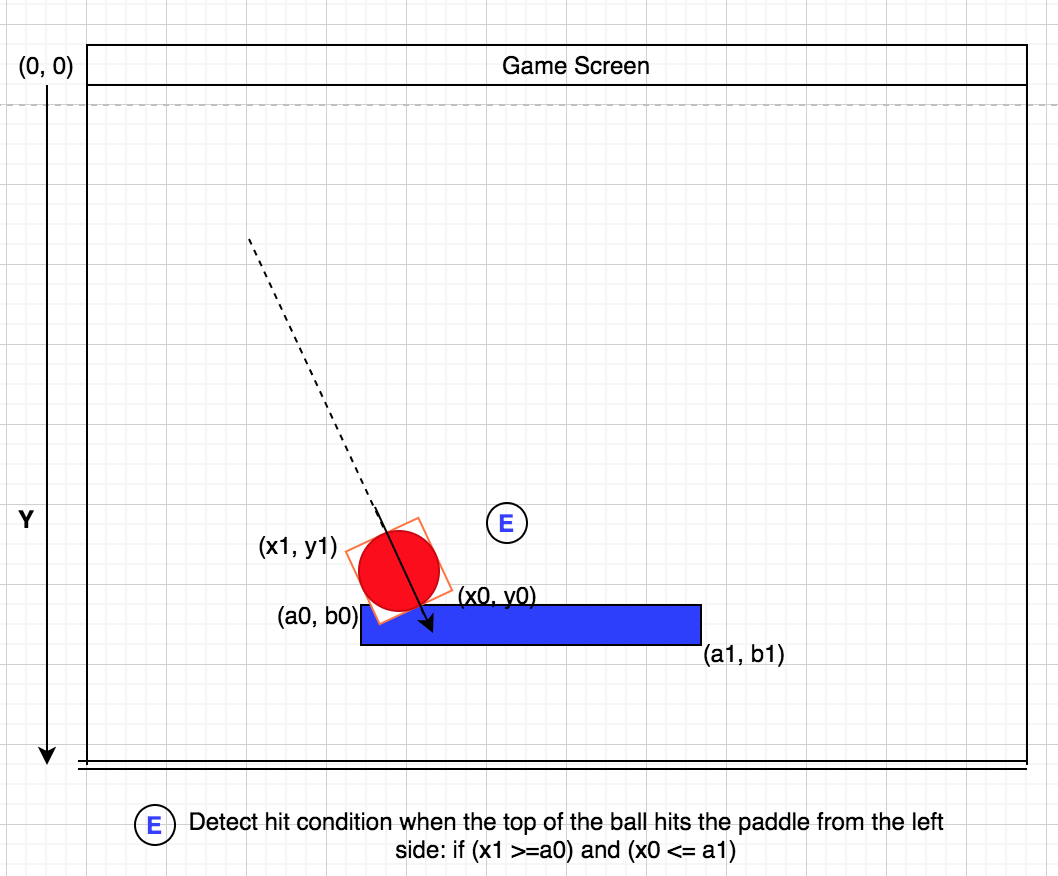


Figure 7

1. When the top of the ball hits the top of the paddle from the right side of the screen. Refer Figure 8

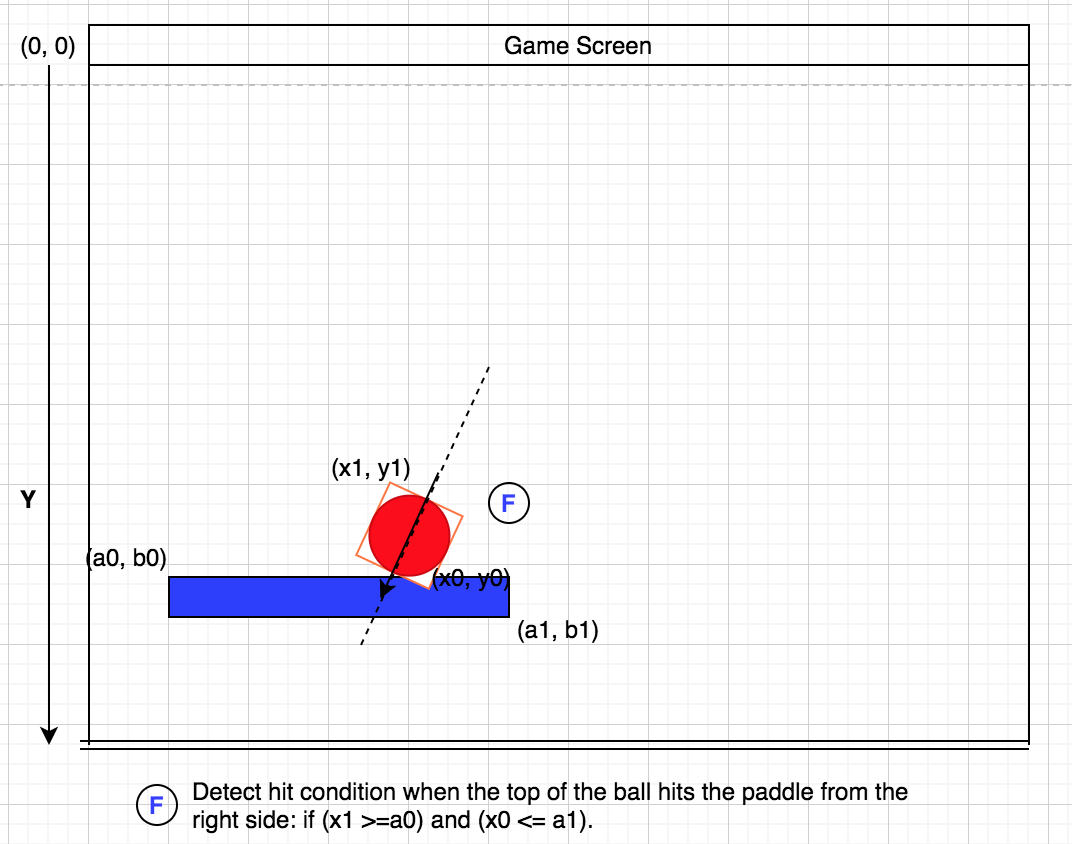


Figure 8

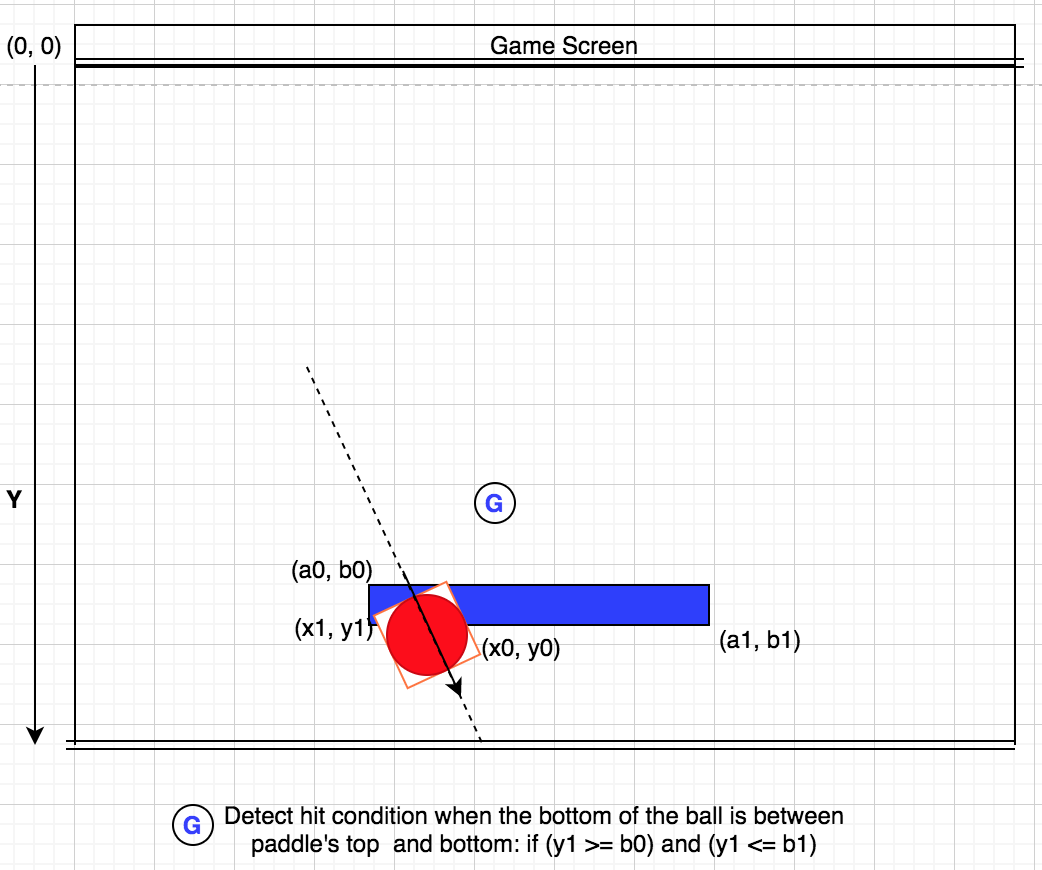
1. When the bottom of the ball lies in between the paddle’s top and bottom. Refer Figure 9.

Figure 9

As Figure 7, 8 and 9 suggest, we will add code for the detecting the hit for the below two conditions:

* (y1 >= b0) and (y1 <= b1)
* (x1 >=a0) and (x0 <= a1)

## Code the Game

In this section, we will add code for the different tasks:

* Create the game canvas
* Create the ball
* Animate the ball
* Create the paddle
* Move the paddle
* Detect collision

### Create the Game Screen

In this section, we will import the Tkinter module and create a Canvas object to build up the game screen. We will also explore the different arguments for changing the width, height and background color of the Canvas object.

1. Locate the IDLE icon on your machine and double click on it to launch the application.
2. Create a new file and save it as “bounceballgame.py”
3. Import the tkinter module

|  |
| --- |
| from tkinter import \* |

1. Create a tkinter object tk = Tk() and add code to give the window a title. This window must be of a fixed size. Using the parameters (0, 0), the size of the window cannot be changed horizontally or vertically.

|  |
| --- |
| tk = Tk()  tk.title ("Game")  tk.resizable(0, 0) |

1. Add code to place the window containing our canvas in front of all the windows.

|  |
| --- |
| tk.wm\_attributes("-topmost", 1) |

1. Create a canvas object with canvas = Canvas() including the parameters as shown below:

|  |
| --- |
| canvas = Canvas (tk, width = 500, height = 400, bd = 0, highlightthickness = 0) |

This line of code says that the canvas must have a width of 500 pixels and height of 400 pixels. For a good game screen design, bd = 0 and highlightthickness = 0 say that the canvas must have no border.

1. Add canvas.pack() to arrange the Tkinter window and pack the canvas screen for the parameters defined in the preceding line.

|  |
| --- |
| canvas.pack() |

1. Add t.update() to initialize the animation defined in the tkinter module.

|  |
| --- |
| tk.update() |

1. Save your program and run your code.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas (tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack() |

Exercise: Try changing the size and boundary thickness of the game screen.

### Create the Ball

In this section, we will create a class for the ball that takes in the parameters for the canvas and the color of the ball that we are going to draw. Using the canvas.create\_oval(x0, y0, x1, y1) we will create the circle shape for the ball. We will then move the ball to the center of the screen using the move() method.

1. Create a class for the ball and define an initialization function that takes in the following parameters:

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, color): |

Note that this class must be defined just below the first line so that the program can reference it while running the final code.

1. Add code to set the object variable “canvas” to the value of the parameter “b\_canvas” defined within the \_\_init\_\_ function.

|  |
| --- |
| self.canvas = b\_canvas |

1. Call the create\_oval function to draw a circle defined using the five parameters: (x, y) coordinates for top-left corner, (x, y) coordinates for the bottom-right corner and the color of the shape.

|  |
| --- |
| self.id = canvas.create\_oval(10, 10, 25, 25, fill=color) |

1. Move the shape created in the preceding line in the middle of the game canvas screen. The object identifier “id” helps the canvas identify which shape to move.

|  |
| --- |
| self.canvas.move(self.id, 245, 100) |

1. Create an object ball that references the class Ball and creates a red ball on the screen.

|  |
| --- |
| ball=Ball(canvas,"red") |

1. Run your final code to display the red ball on your game canvas.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  class Ball:  def \_\_init\_\_(self, b\_canvas, color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=color)  self.canvas.move(self.id, 245, 100)  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas (tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  ball=Ball(canvas,"red") |

Exercise: Try changing the color and size of the ball.

### Animate the Ball

In this section, we will add code to animate the ball by controlling the trajectory of the ball, changing the velocity of the ball as it bounces across horizontal or vertical walls.

#### Make the Ball Move

Before adding some action to the ball, we will make small changes to the code to ensure that it is working. By creating different logical check points, we can manage our code better and troubleshoot it effectively.

1. Add a draw() function in the class Ball as shown below:

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=color)  self.canvas.move(self.id, 245, 100)  def draw(self): |

1. Using the move() method, move the ball from one position to another across the game screen but only 1 pixel vertically up the screen. Thus the x coordinate is 0 and y coordinate is -1

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=color)  self.canvas.move(self.id, 245, 100)  def draw(self):  self.canvas.move(self.id, 0, -1) |

Please note that since the \_\_init\_\_ function saved the canvas parameter as the object variable b\_canvas, it can be referenced in self.canvas along with the move().

1. Save your code and run it. Do you see any changes? If not, play around the value of x and y in the “self.canvas.move(self.id, 0, -1)” and confirm if the position of the ball has changed.
2. Import the time module and add the code below the first line of your program.

|  |
| --- |
| from tkinter import \*  import time |

1. add the code below the last line to instruct the Python interpreter to continuously redraw the screen sleeping intermittently for one hundredth of a second.

|  |
| --- |
| ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

1. Save your code and run it. Your ball is moving but leaves the screen.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  self.x=0  self.y=-1  self.canvas\_height = self.canvas.winfo\_height()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, bg="black", width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

#### Make the Ball Bounce Up and Down

We need to check, on every frame, whether the ball is touching the top edge of the Canvas — if yes, we'll reverse the ball movement so it will start to move in the opposite direction and stay within the visible boundaries. Remembering that the coordinate system starts from the top left, we can come up with something like this:

Cool! Now that our code is running successfully without any errors and the ball is moving, we will add code to make the ball bounce.

1. Create three variables in the \_\_init\_\_function of the class Ball as shown below:

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=color)  self.canvas.move(self.id, 245, 100)  self.x=0  self.y=0  self.canvas\_height = self.canvas.winfo\_height() |

1. Pass the defined variables to the move() method defined in the draw function as shown below:

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y) |

1. Create a variable called “p” and print it. Variable “p” stores the (x, y) coordinates of anything drawn on the canvas. We will use the coords() method to retrieve the (x, y) of the object variable “id”.

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p) |

1. Save the code and run it. The Python Shell window displays an output something like this:

|  |
| --- |
| [255.0, 109.0, 270.0, 124.0] |

These coordinates correspond to the [x0, y0, x1, y1] of the red ball displayed on our screen.

1. Add the code below for implementing bouncing the ball. You may remove the print (p) statement. The print statement was a checkpoint for us to confirm if the code is working successfully.

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=1  if p[3] >=self.canvas\_height:  self.y=-1 |

1. Save your code and run it. The ball should bounce up and down the canvas.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  self.x=0  self.y=-1  self.canvas\_height = self.canvas.winfo\_height()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=1  if p[3] >=self.canvas\_height:  self.y=-1    tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

Exercise: Can you increase the speed of the ball? Hint: Change the value of “self.y”

#### Make the Ball Bounce Left and Right

In this section, we will add code to bounce the ball when it touches the left and right side of the screen.

1. Add a new variable “canvas\_width” to store the width of the canvas.

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  self.x=0  self.y=-1  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width() |

1. Using “canvas\_width”, we will identify if the ball is hitting the right or left side of the screen. Add the code below in the draw() function to add this behavior to the ball:

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=1  if p[3] >=self.canvas\_height:  self.y=-1  if p[0] <= 0:  self.x = 1  if p[2] >= self.canvas\_width:  self.x=-1 |

1. Save your code and run it. The ball should bounce off the screen boundaries.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  self.x=0  self.y=-1  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=1  if p[3] >=self.canvas\_height:  self.y=-1  if p[0] <= 0:  self.x = 1  if p[2] >= self.canvas\_width:  self.x=-1  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

### 

#### Move the Ball in Random Directions

In this section, we will import the random module and randomize the starting (x, y) coordinates of the ball.

1. In the \_\_init\_\_ function, edit the code as shown below:

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width() |

Instead of making the ball always starting at the same x coordinate (“self.x = 0”) on the canvas, we are randomly picking a different location from a list of predefined starting locations defined in the variable list “starts = [-3, -2, -1, 1, 2, 3]”.

We have also changed the value of “self.y” to -3 to increase the speed of the ball.

1. Edit the values of “self.y” to 3 and -3 to make the ball move in the same speed in all directions.

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3 |

1. Save your code and run it. The ball now moves in random directions bouncing off the balls.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  print (p)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

#### Add Speed Variations to the Ball

Ball is moving in random directions while bouncing off balls but moves with the same speed. In real life, the speed of the ball can change as it moves across different boundaries. In this section, we will add code to make the bounce effect more realistic by changing the speed of the ball by a small percentage upon impact with any boundary.

1. In the draw() function, add code to define a variable of data type list “bouncespeed = [-5, -2, -1, 1, 7]” and shuffle it.

|  |
| --- |
| def draw(self):    self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3 |

1. Instead of making the ball always starting at the same speed upon impact, we are randomly changing the speed by a small percentage. Add code to randomly pick a value from the bouncespeed list and apply the change.

|  |
| --- |
| def draw(self):    self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3-bouncespeed[0]  if p[3] >=self.canvas\_height:  self.y=-3-bouncespeed[0]  if p[0] <= 0:  self.x = 3-bouncespeed[0]  if p[2] >= self.canvas\_width:  self.x=-3-bouncespeed[0] |

1. Save your code and run it. The ball now moves in random directions bouncing off the balls with varying speeds.

#### Compare your code

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill=b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()  self.a=2    def draw(self):    self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3-bouncespeed[0]  if p[3] >=self.canvas\_height:  self.y=-3-bouncespeed[0]  if p[0] <= 0:  self.x = 3-bouncespeed[0]  if p[2] >= self.canvas\_width:  self.x=-3-bouncespeed[0]  tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  while 1:  ball.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

### Create the Paddle

In this section, we will use the create\_rectangle function to add a paddle to the screen. Similar to the “class Ball”, we will create a “class Paddle” and add it to the canvas.

1. Create a class for the paddle and define an initialization function that takes in the following parameters:

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill = b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3  class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color): |

Note that this class must be added just below the class Ball.

1. Add code to set the object variable “canvas” to the value of the parameter “p\_canvas” defined within the \_\_init\_\_ function.

|  |
| --- |
| def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas |

1. Call the create\_rectangle function to draw a rectangle defined using the five parameters: (x, y) coordinates for top-left corner, (x, y) coordinates for the bottom-right corner and the color of the shape.

|  |
| --- |
| def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color) |

1. Move the shape created in the preceding line in the bottom section of the game canvas screen. The object identifier “id” helps the canvas identify which shape to move.

|  |
| --- |
| def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300) |

1. Create an object of the “class Paddle” and edit the while loop to call the draw() function for the paddle.

|  |
| --- |
| while 1:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

1. Save your code and run it. A paddle should be added on the screen.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill = b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3  class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)    def draw(self):    tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  paddle=Paddle(canvas, "blue")  while 1:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

### Move the Paddle

In this section, we will add code to move the paddle left by 2 pixels when the left arrow key is pressed and move the paddle right by 2 pixels when the right arrow key is pressed.

1. Create variables in the \_\_init\_\_ function as shown below:

|  |
| --- |
| class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)  self.x = 0  self.canvas\_width = self.canvas.winfo\_width() |

1. Add code to make the paddle move to left and right when the left and right arrow keys are respectively pressed:

|  |
| --- |
| class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)  self.x = 0  self.canvas\_width = self.canvas.winfo\_width()  self.canvas.bind\_all("<KeyPress-Left>", self.turn\_left)  self.canvas.bind\_all("<KeyPress-Right>", self.turn\_right)    def turn\_left(self, evt):  self.x = -2  def turn\_right(self, evt):  self.x= 2 |

Two functions “def turn\_left” and “def turn\_right” capture the key movements and move the paddle in the relevant directions.

1. Create a draw() function in the class Paddle to move the paddle. Using coords() and move() methods, we will then add code to capture the movement of the paddle on the canvas. Unlike the ball, the paddle moves only left and right and thus the y coordinate in the move() will be set to 0.

When it touches the left and right side of the screen, it must stop moving. Thus,

|  |
| --- |
| class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)  self.x = 0  self.canvas\_width = self.canvas.winfo\_width()  self.canvas.bind\_all("<KeyPress-Left>", self.turn\_left)  self.canvas.bind\_all("<KeyPress-Right>", self.turn\_right)    def draw(self):  self.canvas.move(self.id, self.x, 0)  p=self.canvas.coords(self.id)  if p[0] <= 0:  self.x=0  elif p[2] >= self.canvas\_width:  self.x=0  def turn\_left(self, evt):  self.x = -2  def turn\_right(self, evt):  self.x= 2 |

1. Save your code and run it. While the ball is bouncing around the screen, the paddle should be able to move to the left when the left key is pressed and move to the right when the right key is pressed. However, the ball does not detect the paddle and bounce off.

#### Compare your code

The final code at this point looks something like this.

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, b\_color):  self.canvas=b\_canvas  self.id=canvas.create\_oval(10, 10, 25, 25, fill = b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  if p[1] <= 0:  self.y=3  if p[3] >=self.canvas\_height:  self.y=-3  if p[0] <= 0:  self.x = 3  if p[2] >= self.canvas\_width:  self.x=-3  class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)  self.x = 0  self.canvas\_width = self.canvas.winfo\_width()  self.canvas.bind\_all("<KeyPress-Left>", self.turn\_left)  self.canvas.bind\_all("<KeyPress-Right>", self.turn\_right)    def draw(self):  self.canvas.move(self.id, self.x, 0)  p=self.canvas.coords(self.id)  if p[0] <= 0:  self.x=0  elif p[2] >= self.canvas\_width:  self.x=0  def turn\_left(self, evt):  self.x = -2  def turn\_right(self, evt):  self.x= 2    tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  ball=Ball(canvas,"red")  paddle=Paddle(canvas, "blue")  while 1:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

### Detect Collision

1. Update \_\_init\_\_ function to include the paddle as a parameter.

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, i\_paddle, b\_color):  self.canvas=b\_canvas |

1. Add code to assign the paddle parameter (i\_paddle) to the object variable paddle.

|  |
| --- |
| class Ball:  def \_\_init\_\_(self, b\_canvas, i\_paddle, b\_color):  self.canvas=b\_canvas  self.paddle = i\_paddle |

1. Update the ball object to include the paddle as a parameter.

|  |
| --- |
| paddle=Paddle(canvas, "blue")  ball=Ball(canvas, paddle, "red")  while 1:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

1. Create a detect\_hit() function that includes a parameter for passing the ball’s position. In this function, we will add code to detect if the ball has hit the paddle. This function must be added above the draw() in the Class Ball.

|  |
| --- |
| def detect\_hit(self, ballposition):    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id) |

1. Add code in the detect\_hit() function that captures the coordinates of the paddle in a variable (paddle\_position).

|  |
| --- |
| def detect\_hit(self, ballposition):  paddle\_position=self.canvas.coords(self.paddle.id)    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id) |

1. Add code to detect if the ball has hit the paddle. [Refer this section for a more detailed explanation.](#_7hte4chr8uu1)

|  |
| --- |
| def detect\_hit(self, ballposition):  paddle\_position=self.canvas.coords(self.paddle.id)  if ballposition[2] >= paddle\_position[1] and ballposition[0] <= paddle\_position[2]:  if ballposition[3] >= paddle\_position[1] and ballposition[3] <= paddle\_position[3]:  return True  return False  def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id) |

1. Update the draw() in Class Ball to reverse the speed of the ball if it has hit the paddle.

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3-bouncespeed[0]  if p[3] >=self.canvas\_height:  self.y=-3-bouncespeed[0]  if self.detect\_hit(p) == True:  self.y=-3  if p[0] <= 0: |

1. Save your code and run it. Does the paddle detect the ball and bounce it off?

#### Compare your code

|  |
| --- |
| from tkinter import \*  import time  import random  class Ball:  def \_\_init\_\_(self, b\_canvas, i\_paddle, b\_color):  self.canvas=b\_canvas  self.paddle=i\_paddle  self.id=canvas.create\_oval(10, 10, 25, 25, fill = b\_color)  self.canvas.move(self.id, 245, 100)  starts = [-3, -2, -1, 1, 2, 3]  random.shuffle(starts)  self.x=starts[0]  self.y=-3  self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()      def detect\_hit(self, ballposition):  paddle\_position=self.canvas.coords(self.paddle.id)  if ballposition[2] >= paddle\_position[1] and ballposition[0] <= paddle\_position[2]:  if ballposition[3] >= paddle\_position[1] and ballposition[3] <= paddle\_position[3]:  return True  return False    def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3-bouncespeed[0]  if p[3] >=self.canvas\_height:  self.y=-3-bouncespeed[0]  if self.detect\_hit(p) == True:  self.y=-3  if p[0] <= 0:  self.x = 3-bouncespeed[0]  if p[2] >= self.canvas\_width:  self.x=-3-bouncespeed[0]  class Paddle:  def \_\_init\_\_(self, p\_canvas, p\_color):  self.canvas=p\_canvas  self.id=canvas.create\_rectangle(0, 0, 100, 10, fill=p\_color)  self.canvas.move(self.id, 200, 300)  self.x = 0  self.canvas\_width = self.canvas.winfo\_width()  self.canvas.bind\_all("<KeyPress-Left>", self.turn\_left)  self.canvas.bind\_all("<KeyPress-Right>", self.turn\_right)    def draw(self):  self.canvas.move(self.id, self.x, 0)  p=self.canvas.coords(self.id)  if p[0] <= 0:  self.x=0  elif p[2] >= self.canvas\_width:  self.x=0  def turn\_left(self, evt):  self.x =-2  def turn\_right(self, evt):  self.x=2    tk = Tk()  tk.title ("Game")  tk.resizable(0, 0)  tk.wm\_attributes("-topmost", 1)  canvas = Canvas(tk, width = 500, height = 400, bd = 0, highlightthickness = 0)  canvas.pack()  tk.update()  paddle=Paddle(canvas, "blue")  ball=Ball(canvas, paddle, "red")  while 1:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

#### Game Over Condition

Currently, when the ball touches the bottom boundary of the screen, it bounces off. In this section we will add code to implement the game over condition––the game ends when the ball touches the bottom boundary of the screen.

1. Create a variable in the \_\_init\_\_ function that monitors the state of the ball if it has hit the ground. Set it to FALSE.

|  |
| --- |
| self.canvas\_height = self.canvas.winfo\_height()  self.canvas\_width = self.canvas.winfo\_width()  self.hit\_bottom=False |

1. Update the main loop of the program to add this condition. The main loop must continuously make the paddle and the ball move as long as the “self.hit\_bottom” is False.

|  |
| --- |
| while 1:  if ball.hit\_bottom == False:  ball.draw()  paddle.draw()  tk.update\_idletasks()  tk.update()  time.sleep(0.01) |

1. Update the draw() function to implement the game over condition. Instead of bouncing off the ball, we will reset the “self.hit\_bottom” to True.

|  |
| --- |
| def draw(self):  self.canvas.move(self.id, self.x, self.y)  p=self.canvas.coords(self.id)  bouncespeed = [-5, -2, -1, 1, 7]  random.shuffle(bouncespeed)  if p[1] <= 0:  self.y=3-bouncespeed[0]  if p[3] >=self.canvas\_height:  self.hit\_bottom=True  if self.detect\_hit(p) == True:  self.y=-3  if p[0] <= 0:  self.x = 3-bouncespeed[0]  if p[2] >= self.canvas\_width:  self.x=-3-bouncespeed[0] |

1. Save the code and run your program. When the ball touches the bottom of the canvas, the game ends.