-

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

I SEMESTER - R 2017

GE8161– PROBLEM SOLVING AND PYTHON PROGRAMMING LABORATORY

LABORATORY MANUAL

Name : _____

Register No : _____

Section : _____

DHANALAKSHMI COLLEGE OF ENGINEERING

VISION

is committed to provide highly disciplined, c onscientious and enterprising professionals conforming to global standards through value based quality education and training.

MISSION

- To provide competent technical manpower capable of meeting requirements of the industry
- To contribute to the promotion of Academic excellence in pursuit of Technical Education at different levels
- To train the students to sell his brawn and brain to the highest bidder but to never put a price tag on heart and soul

PROGRAM EDUCATIONAL OBJECTIVES

1. FUNDAMENTALS

To impart students with fundamental knowledge in Mathematics, Science and fundamentals of engineering that will mould them to be successful professionals

2. CORE COMPETENCE

To provide students with sound knowledge in engineering and experimental skills to identify complex software problems in industry and to develop practical solutions for them

3. BREADTH

To provide relevant training and experience to bridge the gap between theory and practice this enables them to find solutions for real time problems in industry and organization, and to design products requiring interdisciplinary skills

4. PROFESSIONALISM SKILLS

To bestow students with adequate training and provide opportunities to work as team that will build up their communication skills, individual leadership and supportive qualities, and to enable them to adapt and work in ever changing technologies

5. LIFELONG LEARNING

To develop the ability of students to establish themselves as professionals in Computer Science and Engineering and to create awareness about the need for lifelong learning and pursuing advanced degrees

PROGRAM OUTCOMES

a) To apply the basic knowledge of Mathematics, Science and engineering fundamentals in Computer Science and Engineering field

b) To design and conduct experiments as well as to analyze and interpret data and apply the same in the career

c) To design and develop innovative and creative software applications

d) To understand a complex real world problem and develop an efficient practical solution

e) To create, select and apply appropriate techniques, resources, modern engineering and IT tools

f) To understand their roles as a professionals and give the best to the society

g) To develop a system that will meet expected needs within realistic constraints such as economical, environmental, social, political, ethical, safe and sustainable

h) To communicate effectively and make others understand exactly what they are trying to convey in both verbal and written forms

i) To work in a team as team member or a leader and make unique contributions and work with coordination

j) To exhibit confidence in self-education and ability for lifelong learning

k) To develop and manage projects in multidisciplinary environments

GE 8161 – PROBLEM SOLVING AND PYTHON PROGRAMMING LABORATORY

(Common to all branches of B.E. / B.Tech Programmes)

SYLLABUS

COURSE OBJECTIVES

To write, test, and debug simple Python programs.
To implement Python programs with conditionals and loops.
Use functions for structuring Python programs.
Represent compound data using Python lists, tuples, dictionaries.
Read and write data from/to files in Python.

LIST OF EXPERIMENTS:

1.Compute the GCD of two numbers.
2. Find the square root of a number (Newton's method)
3. Exponentiation (power of a number)
4. Find the maximum of a list of numbers
5. Linear search and Binary search
6. Selection sort, Insertion sort
7. Merge sort
8. First n prime numbers
9. Multiply matrices
10. Programs that take command line arguments (word count)
11. Find the most frequent words in a text read from a file
12. Simulate elliptical orbits in Pygame
13.Simulate bouncing ball in Pygame

COURSE OUTCOMES

Upon completion of the course, students will be able to:
Write, test, and debug simple Python programs.
Implement Python programs with conditionals and loops.
Develop Python programs step-wise by defining functions and calling them.
Use Python lists, tuples, dictionaries for representing compound data.
Read and write data from/to files in Python.

GE 8161 – PROBLEM SOLVING AND PYTHON PROGRAMMING LABORATORY
CONTENTS

Ex. No. :01 Date:	GCD OF TWO NUMBERS

Aim:

To write a Python program to find GCD of two numbers.

Algorithm:

1. Define a function named computeGCD()
2. Find the smallest among the two inputs x and y
3. Perform the following step till smaller+1

   Check if ((x % i == 0) and (y % i == 0)), then assign GCD=i
4. Print the value of gcd

Program:

```
def computeGCD(x, y):
   if x > y:
      smaller = y
   else:
      smaller = x
   for i in range(1, smaller+1):
      if((x % i == 0) and (y % i == 0)):
         gcd = i

   return gcd

num1 = 54
num2 = 24

# take input from the user
# num1 = int(input("Enter first number: "))
# num2 = int(input("Enter second number: "))

print("The GCD. of", num1,"and", num2,"is", computeGCD(num1, num2))
```

Sample output :

The GCD of 54 and 24 is 6

Result:

Thus the Python program to compute GCD of two numbers is executed successfully and the output is verified.

Ex. No. :02 Date:	SQUARE ROOT OF A NUMBER

Aim:

To write a Python Program to find the square root of a number by Newton's Method.

Algorithm:

1. Define a function named newtonSqrt().
2. Initialize approx as 0.5*n and better as 0.5*(approx.+n/approx.)
3. Use a while loop with a condition better!=approx to perform the following,
   i. Set approx.=better
   ii. Better=0.5*(approx.+n/approx.)
4. Print the value of approx

Program:

```
def newtonSqrt(n):
    approx = 0.5 * n
    better = 0.5 * (approx + n/approx)
    while better != approx:
       approx = better
       better = 0.5 * (approx + n/approx)
    return approx

print('The square root is' ,newtonSqrt(81))
```

Output:

The square root is 9

Result:

Thus the Python program for finding the square root of a given number by Newton's Method is executed successfully and the output is verified.

Ex. No. : 03 Date:	EXPONENTIATION OF A NUMBER

Aim:

To write a Python program to find the exponentiation of a number.

Algorithm:

1. Define a function named power()
2. Read the values of base and exp
3. Use 'if' to check if exp is equal to 1 or not
   i. if exp is equal to 1, then return base
   ii.if exp is not equal to 1, then return (base*power(base,exp-1))
4. Print the result

Program:

```
def power(base,exp):
    if(exp==1):
        return(base)
    if(exp!=1):
        return(base*power(base,exp-1))
base=int(input("Enter base: "))
exp=int(input("Enter exponential value: "))
print("Result:",power(base,exp))
```

Output:

Enter base: 7

Enter exponential value: 2

Result:49

Result:

Thus the Python program to find the exponentiation of a number is executed successfully and the output is verified.

Ex. No. : 04 Date:	FINDING MAXIMUM FROM A LIST OF NUMBERS

Aim:

To write a Python Program to find the maximum from a list of numbers.

Algorithm:

1. Create an empty list named l
2. Read the value of n
3. Read the elements of the list until n
4. Assign l[0] as maxno
5. If l[i]>maxno then set maxno=l[i]
6. Increment i by 1
7. Repeat steps 5-6 until i<n
8. Print the value of maximum number

Program:

```
l=[]
n=int(input("enter the upper limit"))
for i in range(0,n):
  a=int(input("enter the numbers"))
  l.append(a)
maxno=l[0]
for i in range(0,len(l)):
  if l[i]>maxno:
    maxno=l[i]
print("The maximum number is %d"%maxno)
```

Output:

```
Enter the upper limt 3
Enter the numbers 6
Enter the numbers 9
Enter the numbers 90
The maximum number is 90
```


Result:

Thus a Python Program to find the maximum from the given list of numbers is successfully executed and the output is verified.

Ex. No. : 5(a) Date:	LINEAR SEARCH

Aim:

To write a Python Program to perform Linear Search

Algorithm:

1. Read n elements into the list
2. Read the element to be searched
3. If alist[pos]==item, then print the position of the item
4. Else increment the position and repeat step 3 until pos reaches the length of the list

Program:

```
def search(alist,item):
  pos=0
  found=False
  stop=False
  while pos<len(alist) and not found and not stop:
    if alist[pos]==item:
      found=True
      print("element found in position",pos)
    else:
      if alist[pos]>item:
        stop=True
      else:
        pos=pos+1
  return found
a=[]
n=int(input("enter upper limit"))
for i in range(0,n):
  e=int(input("enter the elements"))
  a.append(e)
x=int(input("enter element to search"))
search(a,x)
```

Output:

Enter upper limit 5

Enter the elements 6

Enter the elements 45

Enter the elements 2

Enter the elements 61

Enter the elements 26

Enter element to search 6

Element found in position 1

Result:

Thus the Python Program to perform linear search is executed successfully and the output is verified.

EX. No. : 5(b) Date:	BINARY SEARCH

Aim:

To write a Python Program to perform binary search.

Algorithm:

1. Read the search element
2. Find the middle element in the sorted list
3. Compare the search element with the middle element
   i. if both are matching, print element found
   ii. else then check if the search element is smaller or larger than the middle element
4. If the search element is smaller than the middle element, then repeat steps 2 and 3 for the left sublist of the middle element
5. If the search element is larger than the middle element, then repeat steps 2 and 3 for the right sublist of the middle element
6. Repeat the process until the search element if found in the list
7. If element is not found, loop terminates

Program:

```
def bsearch(alist,item):
 first=0
 last=len(alist)-1
 found=False
 while first<=last and not found:
  mid=(first+last)//2
  if alist[mid]==item:
   found=True
   print("element found in position",mid)
   else:
    if item<alist[mid]:
     last=mid-1
```

```
        else:
          first=mid+mid-1
  return found

a=[]
n=int(input("enter upper limit"))
for i in range(0,n):
  e=int(input("enter the elements"))
  a.append(e)
x=int(input("enter element to search"))
bsearch(a,x)
```

Output:

```
enter upper limit 6
enter the elements 2
enter the elements 3
enter the elements 5
enter the elements 7
enter the elements 14
enter the elements  25
enter element to search 5
element found in position 2
```

Result:

Thus the Python Program to perform binary search is executed successfully and the output is verified

Ex. No. : 6(a) Date:	SELECTION SORT

Aim:

To write a Python Program to perform selection sort.

Algorithm:

1. Create a function named selectionsort
2. Initialise pos=0
3. If alist[location]>alist[pos] then perform the following till i+1,
4. Set pos=location
5. Swap alist[i] and alist[pos]
6. Print the sorted list

Program:

```
def selectionSort(alist):
   for i in range(len(alist)-1,0,-1):
       pos=0
       for location in range(1,i+1):
           if alist[location]>alist[pos]:
               pos= location
       temp = alist[i]
       alist[i] = alist[pos]
       alist[pos] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```

Output:

[17, 20, 26, 31, 44, 54, 55, 77, 93]

Result:

Thus the Python Program to perform selection sort is successfully executed and the output is verified.

Ex. No. : 6(b) Date:	INSERTION SORT

Aim:

To write a Python Program to perform insertion sort.

Algorithm:

1. Create a function named insertionsort
2. Initialise currentvalue=alist[index] and position=index
3. while position>0 and alist[position-1]>currentvalue, perform the following till len(alist)
4. alist[position]=alist[position-1]
5. position = position-1
6. alist[position]=currentvalue
7. Print the sorted list

Program:

```
def insertionSort(alist):
   for index in range(1,len(alist)):
     currentvalue = alist[index]
     position = index
     while position>0 and alist[position-1]>currentvalue:
        alist[position]=alist[position-1]
        position = position-1
     alist[position]=currentvalue
alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

Output:

[17, 20, 26, 31, 44, 54, 55, 77, 93]

Result:

Thus the Python program to perform insertion sort is successfully executed and the output is verified.

Ex. No. : 7 Date:	MERGE SORT

Aim:

To write a Python Program to perform Merge sort.

Algorithm:

1. Create a function named mergesort
2. Find the mid of the list
3. Assign lefthalf = alist[:mid] and righthalf = alist[mid:]
4. Initialise i=j=k=0
5. while i < len(lefthalf) and j < len(righthalf), perform the following
   if lefthalf[i] < righthalf[j]:
   alist[k]=lefthalf[i]
   Increment i
   else
   alist[k]=righthalf[j]
   Increment j
   Increment k
6. while i < len(lefthalf),perform the following
   alist[k]=lefthalf[i]
   Increment i
   Increment k
7. while j < len(righthalf), perform the following
   alist[k]=righthalf[j]
   Increment j
   Increment k
8. Print the sorted list

Program:

```
def mergeSort(alist):
  # print("Splitting ",alist)
   if len(alist)>1:
       mid = len(alist)//2
       lefthalf = alist[:mid]
       righthalf = alist[mid:]
       mergeSort(lefthalf)
       mergeSort(righthalf)
       i=j=k=0
       while i < len(lefthalf) and j < len(righthalf):
          if lefthalf[i] < righthalf[j]:
             alist[k]=lefthalf[i]
             i=i+1
          else:
             alist[k]=righthalf[j]
             j=j+1
          k=k+1
       while i < len(lefthalf):
          alist[k]=lefthalf[i]
          i=i+1
          k=k+1
       while j < len(righthalf):
          alist[k]=righthalf[j]
          j=j+1
          k=k+1
   #print("Merging ",alist)
alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

Output:

[17, 20, 26, 31, 44, 54, 55, 77, 93]

Result:

Thus the Python program to perform merge sort is successfully executed and the output is verifie

Ex. No. : 8 Date:	FIRST N PRIME NUMBERS

Aim:

To write a Python program to find first n prime numbers.

Algorithm:

1. Read the value of n
2. for num in range(0,n + 1), perform the following
3. if num%i is 0 then break
   else print the value of num
4. Repeat step 3 for i in range(2,num)

Program:

```
n = int(input("Enter the upper limit: "))

print("Prime numbers are")

for num in range(0,n + 1):
  # prime numbers are greater than 1
  if num > 1:
     for i in range(2,num):
        if (num % i) == 0:
           break
     else:
        print(num)
```

Output:

```
Enter the upper limit:100
Prime numbers are
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```


Result:

Thus the Python Program to find the first n prime numbers is executed successfully and the output is verified.

Ex. No. : 9 Date:	MATRIX MULTIPLICATION

Aim:

To write a Python program to multiply matrices.

Algorithm:

1. Define two matrices X and Y
2. Create a resultant matrix named 'result'
3. for i in range(len(X)):
   i. for j in range(len(Y[0])):
      a) for k in range(len(Y))
      b) result[i][j] += X[i][k] * Y[k][j]
4. for r in result, print the value of r

Program:

```
X = [[12,7,3],
   [4 ,5,6],
   [7 ,8,9]]
Y = [[5,8,1,2],
   [6,7,3,0],
   [4,5,9,1]]
result = [[0,0,0,0],
        [0,0,0,0],
        [0,0,0,0]]
for i in range(len(X)):
  for j in range(len(Y[0])):
     for k in range(len(Y)):
        result[i][j] += X[i][k] * Y[k][j]
for r in result:
  print(r)
```

Output:

[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23]

Result:

Thus the Python program to multiply matrices is executed successfully and the output is verified.

Ex. No. :10 Date:	COMMAND LINE ARGUMENTS(WORD COUNT)

Aim:

To write a Python program for command line arguments.

Algorithm:

1. Add arguments to find the words and lines
2. Add the filename as argument
3. Parse the arguments to get the values
4. Format and print the words and lines

Program:

```
parser=argparse.ArgumentParser()
parser.add_argument('--words','-w',action='store_true')
parser.add_argument('--lines','-l',action='store_true')
parser.add_argument('filename')
args=parser.parse_args()
if args.words:
  print("words {}".format(print_words(args.filename))
elif args.lines:
  print("lines {}".format(print_lines(args.filename))
else:
  print ("words {}".format(print_words(args.filename))
  print("lines {}".format(print_lines(args.filename))
```

Output:

python main.py –i input.txt
words 23
lines 1

Result:
Thus the Python program for command line arguments is executed successfully and the output is verified.

Ex. No. : 11 Date:	FINDING MOST FREQUENT WORDS IN A TEXT READ FROM A FILE

Aim:

To write a Python program to find the most frequent words in a text read from a file.

Algorithm:

1. Read the filename
2. Open the file in read mode
3. Read each line from the file to count the lowers and words
4. Read each line from the file to replace the punctuations
5. Split each line into words and count them
6. Print the words and counts

Program:

```
def main():
    filename=raw_input("enter the file").strip()
    infile=open(filename,"r")
    wordcounts={}
    for line in infile:
            processLine(line.lower(),wordcounts)
    pairs=list(wordcounts.items())
    items=[[x,y] for (y,x) in pairs]
    items.sort()
    for i in range(len(items)-1,len(items)-11,-1):
        print(items[i][1]+"\t"+str(items[i][0]))
def processLine(line,wordcounts):
    line=replacePunctuations(line)
    words=line.split()
    for word in words:
        if word in wordcounts:
            wordcounts[word]+=1
        else:
            wordcounts[word]=1
def replacePunctuations(line):
    for ch in line:
        if ch in "~@#$%^&*()_-+=<>?/.,:;!{}[]"":
            line=line.replace(ch," ")
    return line
main()
```

Output:

Enter a filename:a.txt

Hi 1

How 1

Are 1

You 1

Result:

Thus the Python program to find the most frequent words in a text read from a file is executed successfully and the output is verified.

Ex. No. : 12 Date:	SIMULATE ELLIPTICAL ORBITS IN PYGAME

Aim:

To write a Python program to simulate elliptical orbits in Pygame.

Algorithm:

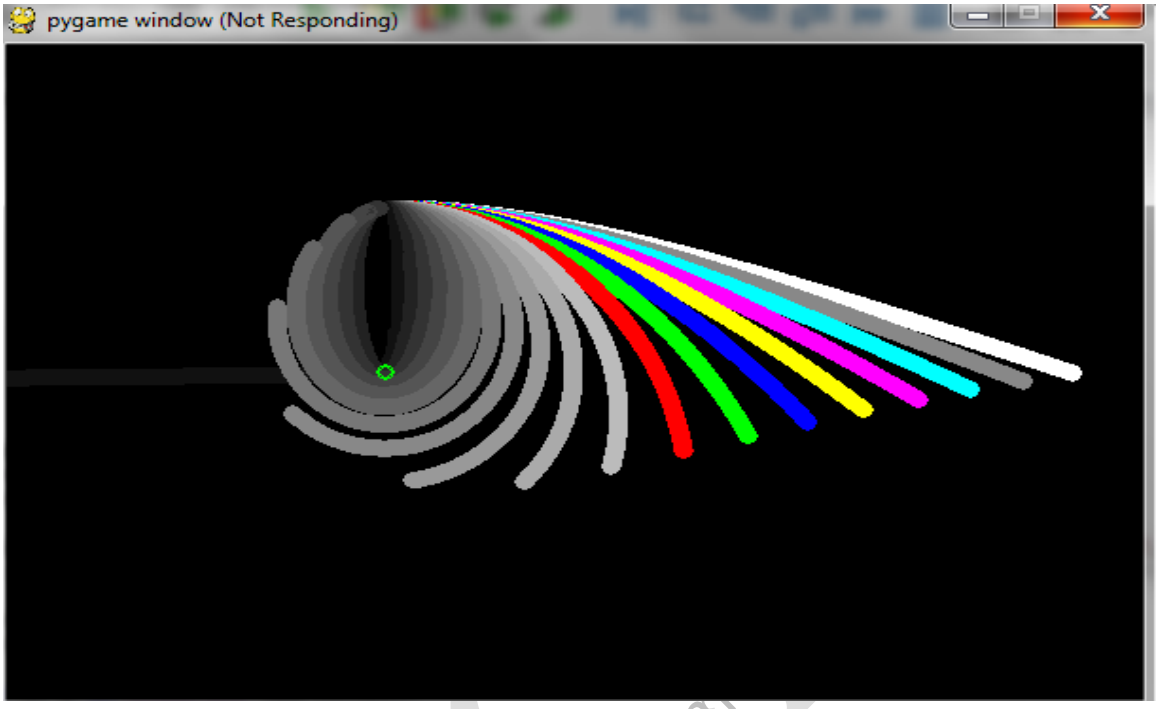
1. Import the required packages
2. Set up the colours for the elliptical orbits
3. Define the parameters to simulate elliptical orbits
4. Display the created orbits

Program:

```
import math
import random
import pygame
class Particle ():
    def __init__ (self, x, y, colour=0x000000):
        self.x = x
        self.y = y
        self.vx = 0
        self.vy = 0
        self.colour = colour
    def apply_gravity (self, target):
        dsqd = (self.x - target.x) ** 2 + (self.y - target.y) ** 2
        #g = G*m/dsqd * normalized (self - target)
        if dsqd == 0:
            return
        self.vx += -1 / dsqd * (self.x - target.x) / dsqd ** 0.5
        self.vy += -1 / dsqd * (self.y - target.y) / dsqd ** 0.5
    def update (self):
        self.x += self.vx
        self.y += self.vy
pygame.init()
window = pygame.display.set_mode ((600, 400))
main_surface = pygame.Surface ((600, 400))
colours = [0x000000, 0x111111, 0x222222, 0x333333, 0x444444, 0x555555, 0x666666,
0x777777, 0x888888, 0x999999, 0xaaaaaa, 0xbbbbbb] + [0xFF0000, 0x00FF00, 0x0000FF,
0xFFFF00, 0xFF00FF, 0x00FFFF, 0x888888, 0xFFFFFF, 0x808000, 0x008080, 0x800080,
0x800000]
#colours = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF, 0x888888,
```

```
0xFFFFFF, 0x808000, 0x008080, 0x800080, 0x800000]
particles = [Particle (200, 100, colours [i]) for i in range (20)]
earth = Particle (200, 200)
for i, p in enumerate (particles):
    p.vx = i / 100
while (True):
#    main_surface.fill(0x000000)
    pygame.draw.circle (main_surface, 0x00FF00, (earth.x, earth.y), 5, 2)

    for p in particles:
        p.apply_gravity (earth)
        p.update ()
        pygame.draw.circle (main_surface, p.colour, (int (p.x), int (p.y)), 5, 2)
    window.blit(main_surface, (0, 0))
    pygame.display.flip()
```

Output:



Result:

Thus the Python Program to simulate elliptical orbits using Pygame is executed successfully and the output is verified.

Ex. No. : 13 Date:	SIMULATE BOUNCING BALL USING PYGAME

Aim:

To write a Python program to bouncing ball in Pygame.

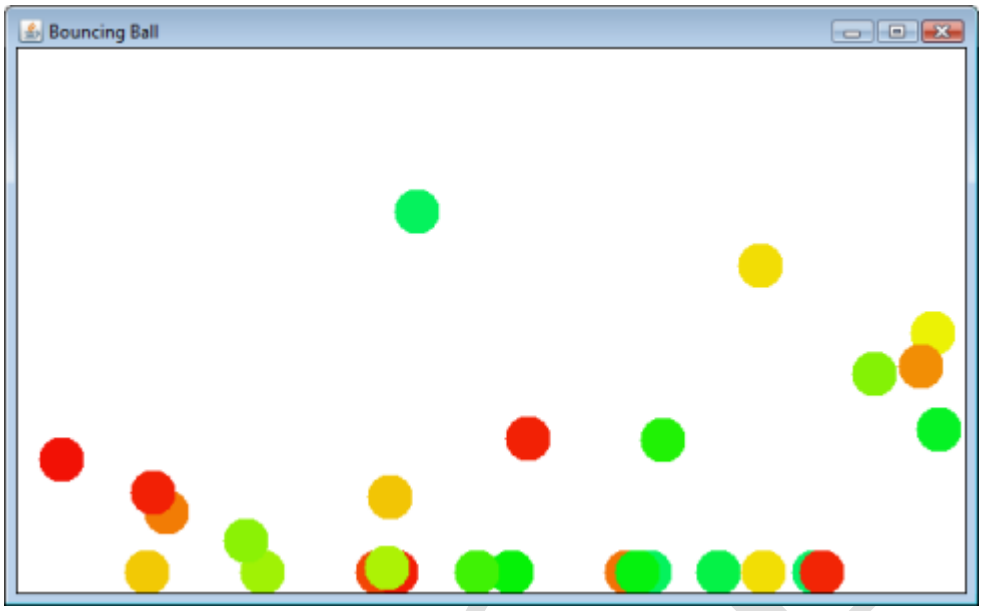
Algorithm:

1. Import the required packages
2. Define the required variables
3. Define the screen space to display the bouncing balls in that space

Program:

```
import sys
import pygame
pygame.init()

size = width, height = 320, 240
speed = [2, 2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)

ball = pygame.image.load('C:\\Users\\admin\\Desktop//ball.jpg')
ballrect = ball.get_rect()

while 1:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()

    ballrect = ballrect.move(speed)
    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]
    if ballrect.top < 0 or ballrect.bottom > height:
        speed[1] = -speed[1]

    screen.fill(black)
    screen.blit(ball, ballrect)
    pygame.display.flip()
```

Output:



Result:

Thus the Python Program to simulate bouncing ball using Pygame is executed successfully and the output is verified.

VIVAVOCE QUESTIONS:

1. What it the syntax of print function?
2. What is the usage of input function?
3. Define a variable.
4. What is type conversion?
5. Mention the data types in Python
6. What are the attributes of the complex datatype?
7. Mention a few escape sequences.
8. Define an expression
9. What is the usage of ** operator in Python?
10. Give the syntax of if else statement.
11. Give the syntax of for statement.
12. How is range function used in for?
13. Give the syntax of while statement.
14. What are multi way if statements?
15. How is random numbers generated?
16. Define a function.
17. Give the syntax of function.
18. What are the types of arguments in function.?
19. What is a recursive function?
20. What are anonymous functions?
21. What are default arguments?
22. What are variable length arguments?
23. What are keyword arguments?
24. Mention the use of map().
25. Mention the use of filter().
26. Mention the use of reduce().
27. Define a string.
28. How is string slicing done?
29. What is the usage of repetition operator?
30. How is string concatenation done using + operator>
31. Mention some string methods
32. How is length of a string found?
33. How is a string converted to its upper case?
34. `Differentiate isalpha() and isdigit().
35. What is the use of split()?
36. Define a file.
37. Give the syntax for opening a file.
38. Give the syntax for closing a file.
39. How is reading of file done?
40. How is writing of file done?
41. What is a list?
42. Lists are mutable-Justify.
43. How is a list created?
44. How can a list be sorted?
45. How are elements appended to the list?
46. How is insert() used in list?

47. What is the usage of pop() in list?
48. Define a tuple.
49. Are tuples mutable or immutable?
50. Mention the use of return statement.
51. What is a Boolean function?
52. How is main function defined?
53. What is a dictionary?
54. How are tuples created?
55. How is a dictionary created?
56. How to print the keys of a dictionary?
57. How to print the values of a dictionary?
58. How is del statement used?
59. Can tuple elements be deleted?
60. What is Python interpreter?
61. Why is Python called an interpreted language?
62. Mention some features of Python
63. What is Python IDLE?
64. Mention some rules for naming an identifier in Python.
65. Give points about Python Numbers.
66. What is bool datatype?
67. Give examples of mathematical functions.
68. What is string formatting operator?
69. Mention about membership operators inPython.
70. How is expression evaluated in Python?
71. What are the loop control statements in Python?
72. What is the use of break statement?
73. What is the use of continue statement?
74. What is the use of pass statement?
75. What is assert statement?
76. Differentiate fruitful function s and void functions.
77. What are required arguments ?
78. Differentiate pass by value and pass by reference.
79. Mention few advantages of function.
80. How is lambda function used?
81. What is a local variable?
82. What are global variables?
83. What are Python decorators?
84. Are strings mutable or immutable?
85. What is join()?
86. What is replace() method?
87. What is list comprehension?
88. Define multidimensional list.
89. How to create lists using range()?
90. What is swapcase() method?
91. What is linear search?
92. How is binary search done?

93. How is merge sort performed?
94. What is sorting?
95. How is insertion sort done?
96. How is selection sort done?
97. What are command line arguments?
98. Name some built in functions with dictionary.
99. What is an exception?
100. How is exception handled in python?

LIST OF MINI PROJECTS

1.ATM Generation

2.Dice Rolling simulator

3.Library Management System

4.Restaurant Management System

5.Calendar Application

6.Guessing Birthdays

7.Loan Calculator

8.Text Animation

9. Spell Checker

10.Data Retrieval from Web