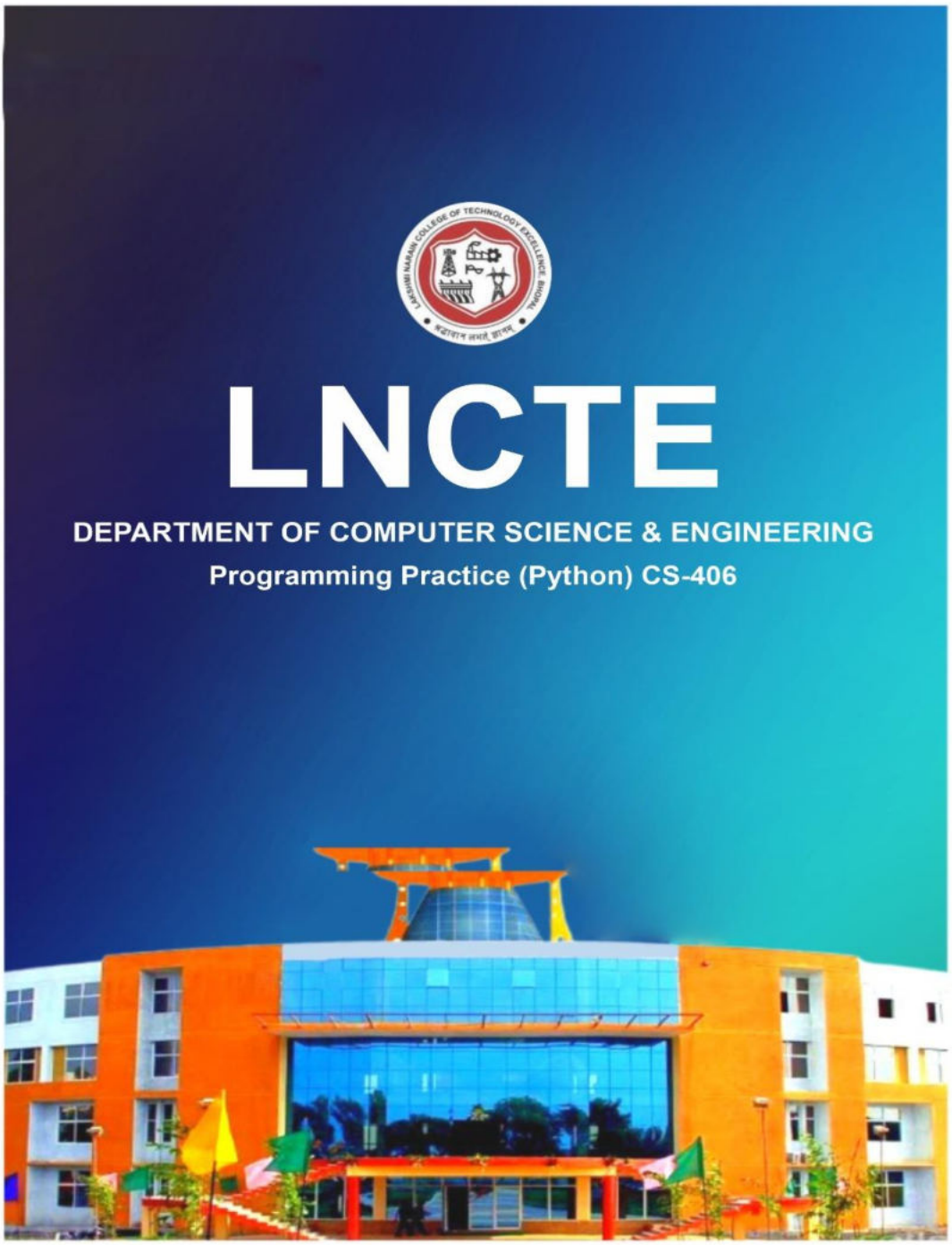
LNCTE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Programming Practice (Python) CS-406

Lakshmi Narain College of Technology Excellence

Department of Computer Science and Engineering

LAB MANUAL

Subject Name: **PROGRAMMING PRACTICES (PYTHON)**

Course Code: CS-406

Course B.Tech.

Sem: IVth

Session: Jan-June 2023-24

Prepared By:

TABLE OF CONTENT

Vision and Mission of the Institute

Vision of the institute

To become a pioneer institute in technical education and innovations to build competent technocrats and leaders for the nation.

Mission of the institute

M1. To enhance the academic environment with innovative teaching learning processes and modern tools.

M2. To Practice and nurture high standards of human values, transparency and accountability.

M3. To collaborate with other academic and research institutes as well as industries in order to strengthen education and research.

M4. To uphold skill development for employability and entrepreneurship for interdisciplinary research and innovations.

Vision and Mission of the Department

VISION OF THE DEPARTMENT

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research &computing, Software product development and entrepreneurship.

MISSION OF THE DEPARTMENT

M1: To offer academic program with state of art curriculum having flexibility for accommodating the latest developments in the areas of Computer science engineering

M2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering.

M3: To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions

Course Outcome: CS-406 PROGRAMMING PRACTICES (PYTHON)

CO1	Analyse basic features of python and compare it with other programming language.
CO2	Implement primitive and derived data structures with python.
CO3	Implement structural and functional programming concept with python.
CO4	Implement object oriented programming concept with python.
CO5	Illustrate standard libraries of python.

Course Articulation Matrix

Course Articulation Matrix

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO406.1	3	3	-	-	-	2	3	-	1	1	1	3
CO406.2	3	3	3	1	-	-	-	-	1	1	1	3
CO406.3	3	3	3	1	-	-	-	-	1	1	1	3
CO406.4	3	3	-	-	2	-	2	-	1	1	1	3
CO406.5	3	3	-	-	2	2	2	-	1	1	1	3
	3	**3**	**2**	**1**	**2**	**2**	**2**	**1**	**2**	**1**	**1**	**3**

Program Outcomes as defined by NBA (PO)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of

technological change.

Program Specific Outcomes (PSO)

A graduate of Computer Science and Engineering Program will develop.

PSO1: An ability to apply technical knowledge of computer science and engineering fundamentals to become employable in industry.

PSO2: An ability to develop programming skills using modern software tools and techniques.

PSO3: An ability to develop real time projects for problem solving of domains such as Machine Learning, Cyber security, block chain and big data.

PSO4: An ability to grab research, higher studies and entrepreneurship opportunities towards society with moral values and ethics.

Program Educational Objectives (PEO):

PEO 1: Evolve as globally competent computer professionals, researchers and entrepreneurs possessing collaborative and leadership skills, for developing innovative solutions in multidisciplinary domains.

PEO 2: Excel as socially committed computer engineers having mutual respect, effective communication skills, high ethical values and empathy for the needs of society.

PEO 3: Involve in lifelong learning to foster the sustainable development in the emerging areas of technology.

PROGRAMMING PRACTICES (PYTHON) (CS-406)

List of Program to be performed:

1	To write a Python program to find GCD of two numbers.	CO1
2	To write a Python Program to find the square root of a number by Newton's Method	CO1
3	To write a Python program to find the exponentiation of a number.	CO1
4	To write a Python Program to find the maximum from a list of numbers.	CO2
5	Write a python program to implement tuple, dictionary and set.	CO2
6	To write a Python program to check the given number is prime numbers.	CO3
7	write a program to implement that the given no is even or odd, in python.	CO3
8	Write a program to implement Write a function to calculate simple interest.	CO3
9	Implement recursion for factorial calculation and Fibonacci series.	CO3
10	Write a program to implement class and object in python.	CO4
11	Write a program to implement inheritance in python.	CO4
12	Write a program to implement overriding in python.	CO4
13	To write a Python program to simulate elliptical orbits in Pygame.	CO5
14	To write a Python program to bouncing ball in Pygame.	CO5

Python Programming.

Theory:- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Prerequisites

You should have a basic understanding of Computer Programming terminologies. A basic understanding of any of the programming languages is a plus.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

First Python Program:

1. **Open notepad and type following program**

   Print ("Hello World")

2. **Save above program with name.py**
3. **Open command prompt and change path to python program location**
4. **Type "*python name.py" (without quotes) to* run the program.**

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Python Variables: Declare, Concatenate, Global & Local

What is a Variable in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

```
a=100
print a
```

Python 1 Example

```
# Declare a variable and
initialize itf = 0
print f
# re-declaring the variable
worksf = 'guru99'
print f
```

Python 2 Example

```
# Declare a variable and
initialize itf = 0
print(f)
# re-declaring the variable
worksf = 'guru99'
print(f)
```

List of some different variable types

```
x = 123                     # integer
x = 123L                    # long integer
x = 3.14                    # double float
x = "hello"                 # string
x = [0,1,2]                 # list
x = (0,1,2)                 # tuple
x = open('hello.py', 'r')   # file
```

Constants

A constant is a type of variable whose value cannot be changed. It is helpful to think of constantsas containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannotbe replaced once place inside the bag.

Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means anew file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example 3: Declaring and assigning value to a constant

Create a constant.py

```
1. PI = 3.14
2. GRAVITY
   =
   9.8
```

Create a main.py

```
1. import constant
2.
3. print(constant.PI)
4. print(constant.GRAVIT
   Y)
```

When you run the program, the output will be:

3.14
9.8z

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

In-bulit Functions in Python:

1. **Strip():**Removes all leading whitespace in string.
2. **len(string)**:Returns the length of the string
3. **upper()**:Converts lowercase letters in string to uppercase.
4. Lower(): Vice versa
5. **split(str="", num=string.count(str))**:Splits string according to delimiter str (space if not provided)and returns list of substrings; split into at most num substrings if given.
6. **replace(old, new [, max])**:Replaces all occurrences of old in string with new or at most maxoccurrences if max given.
7. **find(str, beg=0 end=len(string))**:Determine if str occurs in string or in a substring of string ifstarting index beg and ending index end are given returns index if found and -1 otherwise.

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages −

Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

Python programming language provides following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	if statements

An if statement consists of a boolean expression followed by one or more statements.

2	if...else statements

An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.

3	nested if statements

You can use one if or else if statement inside another if or else ifstatement(s).

Experiment no. 1

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

To find the Greatest Common Divisor (GCD) of two numbers. Here's a simple Python program to achieve that:

```
def find_gcd
    (x, y):
    while y:
      x, y = y, x
    % y return x

num1 = 24
num2 = 36
gcd = find_gcd (num1, num2)
print (f"The GCD of {num1} and {num2} is {gcd}")
```

Output-

The GCD of 24 and 36 is 12

Experiment no. 2

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

Newton's method is an iterative method for finding the approximate roots of a real-valued function. For finding the square root of a number, the function we are interested in finding the root of is:

$f(x)=x^2-\text{number}$.

Solution:

```
def newton_sqrt(number,
    epsilon=1e-10): guess =
    number
    while abs(guess * guess - number)
      > epsilon: guess = (guess +
      number / guess) / 2
    return guess

num = 25
sqrt_result = newton_sqrt(num)
print(f"The square root of {num} is approximately {sqrt_result}")
```

Output-

The square root of 25 is approximately 5.0

Experiment no. 3

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

Python program to calculate the exponentiation of a number using the ** **pow()** operator or by using the function. Here's a simple example using both methods:

```
# Method 1: Using the ** operator
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))
result = base ** exponent
print(f"{base} raised to the power of {exponent} is equal to {result}")

# Method 2: Using the pow() function
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))
result = pow(base, exponent)
print(f"{base} raised to the power of {exponent} is equal to {result}")
```

We then calculate the exponentiation using the ** operator in the first method and the pow() function in the second method.

For example:

```
def exponentiation(base,
  exponent): result = base **
  exponent
  return result
base_num = 2
exp_num = 3
result = exponentiation(base_num, exp_num)
print(f"{base_num} raised to the power of {exp_num} is {result}")
Output
2 raised to the power of 3 is 8
```

Experiment no. 4

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

To find the maximum from a list of numbers without using the **max()**

```
function def
  find_maximum(num
  bers): if not
  numbers:
    return None
maximum = numbers[0] # Initialize maximum to
  the first element for num in numbers:
    if num >
      maximum:
      maximum =
      num
  return maximum
# Example list of numbers
numbers = [int(x) for x in input("Enter the numbers separated
by spaces: ").split()] # Find the maximum number
maximum =
find_maximum(numbers) #
Output the maximum
number
if maximum is not None:
  print("The maximum number in the list is:",
maximum) else:
  print("The list is
empty.") In this
program:
We define a function find_maximum that takes a list of
numbers as input. We initialize the maximum value to
```

the first element of the list.
We iterate through the list and update the maximum value if we encounter a larger number. Finally, we print out the maximum number found in the list.

FOR EXAMPLE:

```
numbers = [10, 5, 8,
20, 15] max_num =
max(numbers)
print(f"The maximum number in the list is: {max_num}")
```

Output

```
The maximum number in the list is: 20
```

Experiment no. 5

Write a python program to implement tuple, dictionary and set.

Python program that demonstrates the implementation of a tuple, dictionary, and set:

```
# Tuple example my_tuple
= (1, 2, 3, 4, 5)
print("Tuple:", my_tuple)

# Dictionary example my_dict
= {'a': 1, 'b': 2, 'c': 3}
print("Dictionary:", my_dict)

# Set example
my_set = {1, 2, 3, 4, 5}
print("Set:", my_set)
```

In this program:

We define a tuple named my_tuple containing some integers.
We define a dictionary named my_dict containing key-value pairs. We define a set named my_set containing some unique integers.
You can run this program to see how tuples, dictionaries, and sets work in Python.

FOR EXAMPLE:

```
# Tuple
my_tuple = (1,
2, 3) #
Dictionary
my_dict = {'key1': 'value1',
'key2': 'value2'} # Set
my_set = {4, 5, 6}
```

```
print(f"Tuple: {my_tuple}\nDictionary: {my_dict}\nSet: {my_set}")
```

Output

```
Tuple: (1, 2, 3)
Dictionary: {'key1': 'value1', 'key2':
'value2'} Set: {4, 5, 6}
```

Experiment no. 6

To write a Python program to check the given number is prime numbers.

```
def
  is_prime(nu
  m): if num
  > 1:
    for i in range(2,
      int(num**0.5) + 1): if
      (num % i) == 0:
        return f"{num} is not a prime number"
    else:
      return f"{num} is a prime number"
  else:
    return f"{num} is not a prime number"

num_to_check = 17
result =
is_prime(num_to_check)
print(result)
```

Output

```
17 is a prime number
```

Experiment no. 7

write a program to implement that the given no is even or odd, in python.

```
def
  check_even_odd(
  num): if num % 2
  == 0:
    return f"{num} is an even
  number" else:
    return f"{num} is an odd number"

num_to_check = 25
result =
check_even_odd(num_to_check)
print(result)
```

Output

25 is an odd number

Experiment no. 8

Write a program to implement a function to calculate simple interest.

```
def
    calculate_simple_interest(principal
    , rate, time): interest = (principal *
    rate * time) / 100
    return interest

principal_amount = 1000
rate_of_interest = 2.5
time_period = 3
simple_interest = calculate_simple_interest(principal_amount,
rate_of_interest, time_period) print(f"The simple interest is:
{simple_interest}")
```

Output

```
The simple interest is: 75.0
```

Experiment no. 9

Implement recursion for factorial calculation and Fibonacci series.

```
# Factorial using
recursion def
factorial_recursive(n
):
  if n == 0 or n ==
    1: return 1
  else:
    return n * factorial_recursive(n - 1)

# Fibonacci series using
recursion def
fibonacci_recursive(n):
  if n <=
    1:
    return
    n
  else:
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

num_for_factorial = 5
```

```
num_for_fibonacci = 6
factorial_result =
factorial_recursive(num_for_factorial)
fibonacci_result =
fibonacci_recursive(num_for_fibonacci)
print(f"The factorial of {num_for_factorial} is:
{factorial_result}")
print(f"The Fibonacci series up to {num_for_fibonacci} terms is: {fibonacci_result}")
```

Output

```
The factorial of 5 is: 120
The Fibonacci series up to 6 terms is: 8
```

Experiment no. 10

Write a program to implement class and object in python.

```
class MyCl ass:
    def _init_(self,
      name):
      self.name =
      name

    def display_name(self):
      print(f"My name is {self.name}")

  # Creating an object of
  MyClass obj =
  MyClass("John")
  obj.display_name()
```

Output

My name is John

Experiment No. 11

Write a program to implement inheritance in python.

Solution

```
class Animal:
  def speak(self):
    print("Animal
    speaks")
class
  Dog(Animal
  ): def
  bark(self):
    print("Dog barks")

# Creating an object of
Dog my_dog = Dog()
my_dog.speak() # Inherits speak method from
Animal class my_dog.bark()
```

Output

```
Animal
speaks Dog
barks
```

Experiment no. 12

Write a program to implement overriding in python.

Solution

```
class Parent:
  def show(self):
    print("Parent's show method")


class
  Child(Parent
  ): def
  show(self):
    print("Child's show method")


# Creating an object of
Child my_child = Child()
my_child.show() # Overrides the show method from Parent class
```

Output

```
Child's show method
```

Experiment no. 13

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

```
# Code for simulating elliptical
orbits in Pygame # (This requires
Pygame library installed)
import
pygame
import sys
# Your Pygame code for elliptical orbits goes here

# Example placeholder code:
pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Elliptical Orbits Simulation")
while True:
  for event in
    pygame.event.get(): if
    event.type ==
    pygame.QUIT:
      pygame.q
      uit()
      sys.exit()

  # Your simulation logic goes here


  pygame.display.flip()
```

Experiment no. 14

To write a Python program to bouncing ball in Pygame.

```
# Code for simulating bouncing

ball in Pygame # (This requires

Pygame library installed) import

pygame

import sys
 # Your Pygame code for bouncing ball goes here
# Example placeholder code:

pygame.init()

screen = pygame.display.set_mode((800, 600))

pygame.display.set_caption("Bouncing Ball Simulation")

while True:

  for event in
    pygame.event.get(): if
    event.type ==

  pygame.QUIT:
      pygame.q
      uit()
      sys.exit()


  # Your simulation logic
  goes here
     pygame.display.flip()
```

Viva voice questions:

1. What is Python? List some popular applications of Python in the world of technology.
2. What are the benefits of using Python language as a tool in the present scenario?
3. Is Python a compiled language or an interpreted language?
4. What is the difference between a Mutable datatype and an Immutable data type?
5. How are arguments passed by value or by reference in Python?
6. What is the difference between a Set and Dictionary?
7. What is List Comprehension? Give an Example.
8. What is a lambda function?
9. What is a pass in Python?
10. How is Exceptional handling done in Python?
11. What is swap case function in Python?
12. . Difference between for loop and while loop in Python.
13. Can we Pass a function as an argument in Python?
14. Is Indentation Required in Python?
15. What is Scope in Python?

VISION OF THE DEPARTMENT

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research &computing, Software product development and entrepreneurship.

MISSION OF THE DEPARTMENT

M1: To offer academic programme with state of art curriculum having flexibility for accommodating the latest developments in the areas of Computer science engineering

M2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering.

M3: To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.