Meerut Institute of Engineering & Technology

Department of Computer Science & Engineering

Dr. A.P.J. Abdul Kalam Technical University, Lucknow

LAB MANUAL

Subject Code: KCS-453

Subject Name: Python Language Programming Lab

INDEX

Vision of the Institute

To be an outstanding institution in the country imparting technical education, providing need based, value based and career based programmes and producing self-reliant, self-sufficient technocrats, capable of meeting new challenges.

Mission of the Institute

To educate young aspirants in various technical fields to fulfill global requirement of human resources by providing sustainable quality education, training and invigorating environment, also molding them into skilled competent and socially responsible citizens who will lead the building of a powerful nation.

Vision of Department

To be an excellent department that imparts value based quality education and uplifts innovative research in the ever-changing field of technology.

Mission of Department

1. To fulfill the requirement of skilled human resources with focus on quality education.
2. To create globally competent and socially responsible technocrats by providing value and need based training.
3. To improve Industry-Institution Interaction and encourage the innovative research activities.

Program Educational Objectives

1. Students will have the successful careers in the field of computer science and allied sectors as an innovative engineer.
2. Students will continue to learn and advance their careers through participation in professional activities, attainment of professional certification and seeking advance studies.
3. Students will be able to demonstrate a commitment to life-long learning.
4. Students will be ready to serve society in any manner and become a responsible and aware citizen.
5. Establishing students in a leadership role in any field.

Program Outcomes

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

1. Ability to apply and analyze computational concepts in the areas related to algorithms, machine learning, cloud computing, web designing and web services.
2. Ability to apply standard practices and methodologies in software development and project management.
3. Ability to employ fundamental concepts and emerging technologies for innovative research activities, carrier opportunities & zeal for higher studies.

Course Outcomes

CO-1	To implement the basic concepts of python programming like math function, Strings, List, Tuple and Dictionary
CO-2	To implement the programs using conditional and loop statements
CO-3	To implement file handling techniques.
CO-4	To implement searching, sorting and merging algorithms.
CO-5	To implement concepts of OOPS.

List of Experiments

Exp. No.	Experiment Name	Course Outcome
1	a) To write a python program that takes in command line arguments as input and print the number of arguments. b) To write a python program find the square root of a number (Newton's method)	CO1
2	a) To write a python program exponentiation (power of a number). b) To write a python program to compute the GCD of two numbers. c) To write a python program first n prime numbers.	CO2
3	To write a python program find the maximum of a list of numbers. To write a python program to perform Matrix Multiplication.	CO1
4	To write a python program to find the most frequent words in a text file.	CO3
5	a) To write a python program linear search. b) To write a python program Binary search	CO4
6	a) To write a python program selection sort. b) To write a python program Insertion sort.	CO4
7	To write a python program merge sort.	CO4
8	To write a python program simulate bouncing ball in Pygame.	CO1
Value added programs		
9.	a) To demonstrate working of classes and objects b) To demonstrate constructors c) To demonstrate class method and static method	**CO5**
10.	a) Concept of polymorphism in python (method overloading and overriding) b) To demonstrate inheritance	**CO5**

Program 1(a)

Object: To write a python program that takes in command line arguments as input and print the number of arguments.

Procedure: Python provides a getopt module that helps you parse command-line options and arguments.

```
$ python test.py arg1 arg2 arg3
```

The Python sys module provides access to any command-line arguments via the sys.argv. This serves two purposes −

- sys.argv is the list of command-line arguments.
- len(sys.argv) is the number of command-line arguments.

Here sys.argv[0] is the program ie. script name.

Command to be executed on command line:

```
$ python test.py arg1 arg2 arg3
```

Result −

```
Number of arguments: 4 arguments.
Argument List: ['test.py', 'arg1', 'arg2', 'arg3']
```

Program-1(b)

Object: Write a Python Program to find the square root of a number by Newton’s Method

Newton’s Method

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.

Input:

```
9
```

Output:

```
3
```

Program 2(a)

Object: 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,
4. then return (base*power(base,exp-1))
5. Print the result.

Input:

```
Enter base: 7

Enter exponential value: 2
```

Output:

```
49
```

Program-2(b)

Object: To write a python program to compute the GCD of two numbers.

Algorithm:

```
1. Define a function named compute GCD()
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
```

Input:

M=12, N=17

Output:

1

Program-2(C)

Object: To write a python program 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)
```

Input:

```
Enter the upper limit: 20
```

Output:

```
Prime numbers are

2 3 5 7 11 13 17 19
```

Program-3(a)

Object: To write a python program find the maximum of 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
```

Input:

list1 = [10, 20, 4, 45, 99]

Output-

```
Max=99
```

Program 3(b)

Object: To write a python program to perform Matrix Multiplication.

Algorithm:

```
1. Define two matrices X and Y
2. Create a resultant matrix named 'result'
3. for i in range(len(X)):
        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
```

Input:

# 3x3 matrix

X = [[12,7,3], [4 ,5,6], [7 ,8,9]]

# 3x4 matrix

Y = [[5,8,1,2], [6,7,3,0], [4,5,9,1]]

Output:

```
[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23
```

Program-4

Object: To write a python program to find the most frequent words in a text file.

Algorithm:

1. Open the file in read mode and assign to fname variable.
2. Initialize count, maxcount to 0.
3. Spilt the file into no. of lines.
4. Spilt the lines into words.
5. Loop through each words and count the occurrences.

Input:

Let us consider you have a text file with contents like this

Hi, friends this program is found in text.

This program works perfectly

Output:

Most frequent words–“program”

Program 5(a)

Object: 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

Input:

list of items is: [5, 7, 10, 12, 15])

Enter item to search: 7

Output:

item found at position:,

Program-5(b)

Object: To write a python program Binary search.

Algorithm:

1. Read n elements into the list
2. Read the element x to be searched
3. Compare x with the middle element.
4. If x matches with middle element, we return the mid index.
5. Else If x is greater than the mid element, then x can only lie **in** right half subarray
    a. After the mid element. So we recur for right half.
6. Else (x is smaller) recur for the left half.

Input:

arr =[2, 3, 4, 10, 40]

x =10

Output:

Element is present at index 3

Program-6(a)

Object: To write a python program selection sort.

Algorithm:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Steps

Step 1 − Set MIN to location 0

Step 2 − Search the minimum element in the list

Step 3 − Swap with value at location MIN

Step 4 − Increment MIN to point to next element

Step 5 − Repeat until list is sorted

Input:

Array=[22, 12, 11, 64, 25]

Output:

Sorted Array: [11, 12, 22, 25, 64]

Program-6(b)

Object: To write a python program Insertion sort.

Algorithm:

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands. Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages: Efficient for (quite) small data sets, much like other quadratic sorting algorithms. More efficient in practice than most other simple quadratic (i.e., O(n2)) algorithms such as selection sort or bubble sort

Steps:

Step 1 − If it is the first element, it is already sorted. return 1;

Step 2 − Pick next element

Step 3 − Compare with all elements in the sorted sub-list

Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Step 5 − Insert the value

Step 6 − Repeat until list is sorted

Input:

Array: 11, 6, 13, 12, 5

Output:

Sorted array is: 5, 6, 11, 12, 13

Program-7

Object: To write a python program Merge sort.

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge () function** is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l,  r)

If r > l

   1. Find the middle point to divide the array into two halves:

        middle m = (l+r)/2

   2. Call mergeSort for first half:

        Call mergeSort(arr, l, m)

   3. Call mergeSort for second half:

        Call mergeSort(arr, m+1, r)

   4. Merge the two halves sorted in step 2 and 3:

        Call merge(arr, l, m, r)
```

Input:

Given array is

12 11 13 5 6 7

Output:

Sorted array is

```
5 6 7 11 12 13
```

Program 8

Object: To write a python program to simulate bouncing ball in Pygame.

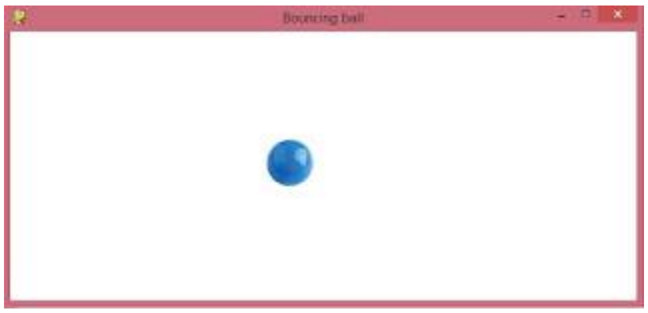
ALGORITHM:

STEP 1: Define the class Ball and initialize the screen background, image and the circle for the ball.

STEP 2: Define the functions for update and for checking the boundary for the ball to hit

STEP 3: Define the main function for the actual bouncing ball simulation

Output:



Value Addition Program-9

Objective: To demonstrate working of classes and objects

Procedure:

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows −

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its *__init__* method accepts.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows −

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

Instead of using the normal statements to access attributes, you can use the following functions

- The getattr(obj, name[, default]) − to access the attribute of object.
- The hasattr(obj,name) − to check if an attribute exists or not.
- The setattr(obj,name,value) − to set an attribute. If attribute does not exist, then it would be created.

- The delattr(obj, name) − to delete an attribute.

```
hasattr(emp1, 'age')    # Returns true if 'age' attribute exists

getattr(emp1, 'age')    # Returns value of 'age' attribute

setattr(emp1, 'age', 8) # Set attribute 'age' at 8

delattr(empl, 'age')    # Delete attribute 'age'
```

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

- __dict__ − Dictionary containing the class's namespace.
- __doc__ − Class documentation string or none, if undefined.
- __name__ − Class name.
- __module__ − Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- __bases__ − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Destroying Objects (Garbage Collection)

Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.

An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

```
a = 40      # Create object <40>

b = a       # Increase ref. count of <40>

c = [b]     # Increase ref. count of <40>


del a       # Decrease ref. count  of <40>

b = 100     # Decrease ref. count of <40>
```

```
c[0] = -1   # Decrease ref. count  of <40>
```

You normally will not notice when the garbage collector destroys an orphaned instance and reclaims its space. But a class can implement the special method *__del__ ()*, called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any non-memory resources used by an instance.

Objective: To demonstrate class method & static method

Procedure:

Class Method

The @classmethod decorator, is a builtin function decorator that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your functiondefinition.
A class method receives the class as implicit first argument, just like an instance method receives the instance
Syntax:

```
class C(object):

   @classmethod

   def fun(cls, arg1, arg2, ...):

      ....
```

fun: function that needs to be converted into a class method

returns: a class method for function.

- A class method is a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances.

Static Method

A static method does not receive an implicit first argument.
Syntax:

```
class C(object):

    @staticmethod

    def fun(arg1, arg2, ...):

        ...
```

returns: a static method for function fun.

- A static method is also a method which is bound to the class and not the object of the class.
- A static method can't access or modify class state.
- It is present in a class because it makes sense for the method to be present in class.

Objective: To demonstrate constructors

Constructors are generally used for instantiating an object.The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created.In Python the __init__() method is called the constructor and is always called when an object is created.

Syntax of constructor declaration:

```
def __init__(self):

    # Body of the constructor
```

Types of constructors:

- **Defaultconstructor: The** default constructor is simple constructor which doesn't accept any arguments.Its definition has only one argument which is a reference to the instance being constructed.
- **Parameterizedconstructor:** constructorwith parameters is known as

parameterized constructor.The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Value Addition Program-10

Objective: To demonstrate inheritance

Class Inheritance:

Instead of starting from scratch, you can create a class by deriving it from a pre-existing class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

Syntax

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name –

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Optional class documentation string'
   class_suite
```

Objective:Concept of polymorphism in python(method overloading and overriding)

Sometimes an object comes in many types or forms. If we have a button, there are many different draw outputs (round button, check button, square button, button with image) but they do share the same logic: onClick(). We access them using the same method. This idea is called *Polymorphism*.

Polymorphism is based on the greek words Poly (many) and morphism (forms). We will create a structure that can take or use many forms of objects.

Overriding Methods

You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

Example:

```
class Bear(object):

   def sound(self):

      print "Groarrr"


class Dog(object):

   def sound(self):

      print "Woof woof!"


def makeSound(animalType):

   animalType.sound()


bearObj = Bear()

dogObj = Dog()
```

```
makeSound(bearObj)

makeSound(dogObj)
```

Output:

```
Groarrr

Woof woof!
```