SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT

OF

MASTER OF COMPUTER APPLICATION

ACADEMIC YEAR: 2025-2026 (ODD SEMESTER)

LAB MANUAL

(REGULATION - 2024)

MC4167 – PYTHON PROGRAMMING LABORATORY

FIRST SEMSTER

MCA – COMPUTER APPLICATIONS

Prepared By

MR. K.MANIRAJ M.Sc.,M.Tech.,(Ph.D)

Assistant Professor

VISION OF THE DEPARTMENT

To educate students with conceptual knowledge and technical skills in the field of Computer Applications with moral and ethical values to achieve excellence in academic, industry, and research- centric environments.

MISSION OF THE DEPARTMENT

1. To bring the most brilliant students and faculty together to understand the strengths and limits of computation, invent next-generation computing systems, and create innovative solutions to real-world problems.

2. To provide conducive environment so as to achieve excellence in teaching-learning, and research and development activities.

3. Deliver knowledge among students through novel pedagogical methods in the varied areas of computer sciences with thrust on applications so as to enable students to undertake research.

4. To facilitate students to nurture skills to practice their professions competently to meet the ever-changing needs of society.

INDEX

MC4167	PYTHON PROGRAMMING LABORATORY	L T P C
		0 0 4 2

LIST OF EXPERIMENTS:

1. Python programming using simple statements and expressions (exchange the values of two variables, circulate the values of n variables, distance between two points).
2. Scientific problems using Conditionals and Iterative loops.
3. Linear search and Binary search
4. Selection sort, Insertion sort
5. Merge sort, Quick Sort
6. Implementing applications using Lists, Tuples.
7. Implementing applications using Sets, Dictionaries.
8. Implementing programs using Functions.
9. Implementing programs using Strings.
10. Implementing programs using written modules and Python Standard Libraries (pandas,numpy, Matplotlib, scipy)
11. Implementing real-time/technical applications using File handling.
12. Implementing real-time/technical applications using Exception handling.
13. Creating and Instantiating classes

Total: 60 Periods

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

HARDWARE/SOFTWARE REQUIREMENTS

1: Processors: Intel Atom® processor Intel®Core™i3 processor2:
Disk space: 1GB.
3: Operating systems: Windows 7,macOS and Linux
4: Python versions:2.7, 3.6, 3.8

COURSE OUTCOMES

At the end of the course, students will be able to

CO1: Apply the Python language syntax including control statements, loops and functions to solve a wide variety of problems in mathematics and science.

CO2: Use the core data structures like lists, dictionaries, tuples and sets in Python to store, process and sort the data.

CO3: Create files and perform read and write operations.

CO4: Illustrate the application of python libraries.

CO5: Handle exceptions and create classes and objects for any real time applications.

CO- PO MATRIX

Course Outcomes	PROGRAM OUTCOMES					
	1	**2**	**3**	**4**	**5**	**6**
CO1	2	1	3	3	2	2
CO2	2	1	3	3	2	2
CO3	1	1	3	2	2	2
CO4	2	1	3	2	2	2
CO5	2	1	3	3	2	3
AVG	1.8	1	3	2.6	2	2.2

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
Total		**60**

Ex.No:1

PYTHON PROGRAMMING USING SIMPLE STATEMENTS AND EXPRESSIONS

Ex.No:1a

AIM:

To exchange the given values of two variables using python.

PRE LAB DISCUSSION:

An expression is a code construct that is evaluated to a value. A code construct is a piece of code. Following are some common expressions:

- An object or a declared variable, such as: 3, Hi, x, [1, 2, 3].
- A computation using operators, such as 3 + 5, x < y < z.
- A function call, such as len("hello"), math.pow(3, 2).
- Functions defined inside types are called methods. A method call is an expression.
- For example: "hello".upper(), [1, 2, 3].pop().

Simple statements

- assignment statement name = expression
- return statement: return expression
- import statement: import module_name

Compound Statements

A compound statement contains multiple statements that usually span multiple lines. Compound statements are used to control program flow or create new data types like functions and classes.

ALGORITHM:

Step 1: Start the program.

Step 2: Get two integer inputs var1 and var2 from the user using the input() function.

Step 3: Declare third variable, c.

Step 4: c=a, a=b, b=c.

Step 5: Print the output swapped values a and b.

Step 6: Stop the program.

PROGRAM:

```
print("Swapping using temporary variable")
a = int(input("a = "))
b = int(input("b = "))
print("Before Swapping")
print("a = ", a)
print("b = ", b)
c = a
a = b
b = c
print("After Swapping")
print("a = ", a)
print("b = ", b)
```

OUTPUT:

Swapping using temporary variable

a = 10

b = 20

Before Swapping

a = 10

b = 20

After Swapping

a = 20

b = 10

RESULT

Thus the program to exchange the given values of two variables using python has been executed successfully.

Ex.No.1b.

CIRCULATE THE VALUES OF N VARIABLES

Aim:

To circulate the given values of n variables using python.

Algorithm:

Step 1: Start the program.

Step 2: Get one integer input no_of_terms from the user using the input() function.

Step 3: Read the value of no_of_terms.

Step 4: Create a list as list1.

Step 5: Validate the range of no_of_terms.

Step 6: Then, get one integer input ele from the user using input() function.

Step 7: Add a single item to the existing list using the append method.

Step 8: Print the circulative values.

Step 9: Stop the program.

PROGRAM:

```
#Circulate the values of n variables
no_of_terms=int(input("Enter number of values : "))
list1=[]
for val in range(0,no_of_terms,1):
    ele=int(input("Enter integer : "))
    list1.append(ele)
#Circulate and display values
print("Circulating the elements of list ",list)
for val in range(0,no_of_terms,1):
    ele=list1.pop(0)
    list1.append(ele)
    print(list1)
```

OUTPUT:

```
Enter number of values : 5
Enter integer : 8
Enter integer : 5
Enter integer : 3
Enter integer : 6
Enter integer : 7
Circulating the elements of list <class 'list'>
[5, 3, 6, 7, 8]
[3, 6, 7, 8, 5]
[6, 7, 8, 5, 3]
[7, 8, 5, 3, 6]
[8, 5, 3, 6, 7]
```

Result:

Thus the program has been executed successfully circulate the given values of n variables using python.

Ex.No:1c

CALCULATE DISTANCE BETWEEN TWO POINTS

AIM:

To calculate distance between two points using python

ALGORITHM:

Step 1: Start the program.

Step 2: Get two integer inputs x1 and x2 for coordinates 1 from the user using the input() function.

Step 3: Then, Get two integer inputs y1 and y2 for coordinates 2.

Step 4: Calculate using the two points (x1,y1) and (x2,y2),the distance between these points is given by the formula.

Step 5: (((x2-x1)**2+((y2-1)**2))**0.5. **Step**

Step 6: Print the distance between values. **Step**

Step 7: Stop the program.

PROGRAM:

```
#Distance between two points
print("Enter coordinates for Point 1 : ")
x1=int(input("enter x1 : "))
x2=int(input("enter x2 : "))
print("Enter coordinates for Point 2 : ")
y1=int(input("enter y1 : "))
y2=int(input("enter y2 : "))
result= ((((x2 - x1 )**2) + ((y2-y1)**2) )**0.5)
print("distance    between",(x1,x2),"and",(y1,y2),"is    :    ",result)
```

OUTPUT

```
Enter coordinates for Point 1 :
enter x1 : 100
enter x2 : 200
Enter coordinates for Point 2 :
enter y1 : 500
enter y2 : 500
distance between (100, 200) and (500, 500) is :  100.0
```

RESULT

Thus the program to calculate distance between two points using python has been executed successfully

Ex.No:2 SCIENTIFIC PROBLEMS USING CONDITIONALS AND ITERATIVE LOOPS

AIM :

To write a Python Program for scientific problems using conditionals and iterative loops

PRE LAB DISCUSSION

Conditional statements

`if`, `elif`, and `else`.

Basic `if` Statement

```
x = 10
if x > 5:
    print("x is greater than 5")
```

`if-else` Statement

```
x = 4
if x > 5:
print("x is greater than 5")
else:
print("x is 5 or less")
```

`if-elif-else` Statement

```
x = 10
if x > 10:
print("x is greater than 10")
elif x == 10:
 print("x is exactly 10")
        else:
print("x is less than 10")
```

2. Iterative Loops

`for` loops and `while` loops

`for` Loop

`case:1`

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
```

```
print(fruit)
```

`case:2`

```
        for i in range(5):
print(i)
```

`while` Loop

```
       count = 0
while count < 5:
print(count)
 count += 1
 # Increment the count to eventually end the loop
```

Using `if` Inside a `for` Loop

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
if number % 2 == 0:
   print(f"{number} is even")
else:
   print(f"{number} is odd")
```

Using `break` and `continue`

```
# Using break
for i in range(10):
if i == 5:
   break
     print(i)
 # Using continue
 for i in range(10):
       if i % 2 == 0:
         continue
     print(i)
# Prints only odd numbers
```

Ex.No:2a **FIBONACCI SERIES**

AIM:

Write a python program to generate Fibonacci series using function.

PRE LAB DISCUSSION:

Fibonacci Sequence is a series of numbers starting with 0 and 1 in which each number, is generated by adding the two preceding numbers. It is a special sequence of numbers that starts from 0 and 1 and then the next terms are the sum of the previous terms and they go up to infinite terms.

ALGORITHM:

Step1:Start
Step2:Get the number of terms
Step3: Check if the number of terms is valid
Step4:If there is only one term, return n1
Step5:If it not generate the fibonacci sequence upto n terms
Step6:End

PROGRAM:

```
nterms = int(input("How many terms? "))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
      print(n1)
      nth = n1 + n2
      # update values
      n1 = n2
      n2 = nth
      count += 1
```

OUTPUT:

How many terms? 5

Fibonacci sequence:

0

1

1

2

3

RESULT:

Thus the program is executed to find the Fibonacci series of a given number and the output is obtained.

Ex.No:2b **AMSTRONG NUMBER**

AIM :

To write a Python program to find the Armstrong number.

PRELAB DISCUSSION:

Given a number x, determine whether the given number is Armstrong number or not. A positive integer of n digits is called an Armstrong number of order n (order is number of digits) if.

abcd... = pow(a,n) + pow(b,n) + pow(c,n) + pow(d,n) + ...

ALGORITHM:

Step1:Start

Step2:Define Function to calculate x raised to the power y

Step3: Calculate order of the number

Step4:Then add the number with the sum

Step5:Define the function isArmstrong() to check the given is armstrong number or not.

Step6:If it the digit is a armstrong number

Step6:If it is not the digit is not a Armstrong number.

PROGRAM/SOURCE CODE:

```
def power(x, y):
    if y == 0:
        return 1
    if y % 2 == 0:
        return power(x, y // 2) * power(x, y // 2)
    return x * power(x, y // 2) * power(x, y // 2)
def order(x):
    n = 0
    while (x != 0):
        n = n + 1
        x = x // 10
    return n
def isArmstrong(x):
    n = order(x)
    temp = x
    sum1 = 0
```

```
    while (temp != 0):
        r = temp % 10
        sum1 = sum1 + power(r, n)
        temp = temp // 10
    return (sum1 == x)
x = 153
print(isArmstrong(x))
x = 1253
print(isArmstrong(x))
```

OUTPUT:

```
True
False
```

RESULT:

Thus the program is executed to find the given number is Armstrong number or not and the output is obtained.

Ex.No:2c **PALINDROME**

AIM:

Write a Python program to reverse the digits of a given number and add them to the original. Repeat this procedure if the sum is not a palindrome.

PRELAB DISCUSSION:

A palindrome is a word, number, or other sequence of characters which reads the same backward as forward, such as madam or race car.

ALGORITHM:

Step1:Start

Step2:Define the function

Step3:Check the position of the digits

Step4:If the end of the string Matches with the first string then given digit is a palindrome

Step5:If it not matches the digit is not a palindrome

Step5:End

PROGRAM:

```
def rev_number(n):
  s = 0
  while True:
    k = str(n)
    if k == k[::-1]:
      break
    else:
      m = int(k[::-1])
      n += m
      s += 1
  return n
rev=int(input("enter num:"))
print(rev_number(rev))
```

OUTPUT:

Enter num: 145

Pallindrome number is

686

Enter num: 144

Pallindrome number is

585

RESULT:

Thus the program is executed to find the palindrome of a given number and the output is obtained

Ex.No:3a **LINEAR SEARCH**

AIM :

To write a python Program to perform linear search

PRELAB DISCUSSION:

Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found. It is the simplest searching algorithm.

ALGORITHM:

Step 1: Start.
Step 2: Read the number of element in the list.
Step 3: Read the number until loop n -1.
Step 4: Then Append the all element in list
Step 5: Go to STEP -3 upto n -1.
Step 6 : Read the searching element from the user
Step 7 : Assign to FALSE flag value
Step 8 : Search the element with using for loop until length of list
Step 9 : If value is found assign the flag value is true
Step10 : Then print the output of founded value an d position.
Step 11 : If value is not found then go to next step
Step 12 : Print the not found statement

PROGRAM :

```
a=[ ]
n=int(input("Enter number of
elements:")) for i in range(1,n+1):
b=int(input("Enter element:"))
a.append(b)
x = int(input("Enter number to search: "))
 found = False
for i in range(len(a)):
if(a[i] = = x):
found = True
print("%d found at%dthposition"%(x,i))
break
if (found==False):
print("%d is not in list"%x)
```

OUTPUT 1:

Enter number of elements:5

Enter element:88

Enter element:11

Enter element:64

Enter element:23

Enter element:89

Enter number to search: 11

11 found at 1th position

OUTPUT 2:

Enter number of elements:5

Enter element:47

Enter element:99

Enter element:21

Enter element:35

Enter element:61

Enter number to search: 50

50 is not in list

RESULT:

Thus the program to perform linear Search is executed and the output is obtained.

Ex.No. 3b. **BINARY SEARCH**

AIM:

To write a python program to perform the binary search.

PRELAB DISCUSSION:

The divide and conquer approach technique is followed by the recursive method. In this method, a function is called itself again and again until it found an element in the list.
A set of statements is repeated multiple times to find an element's index position in the iterative method. The while loop is used for accomplish this task.
Binary search is more effective than the linear search because we don't need to search each list index. The list must be sorted to achieve the binary search algorithm.

ALGORITHM:

Step:1 - mid = (starting index + last index) / 2
Step:2 - If starting index > last index
Then, Print "Element not found"
Exit
Else if element >arr[mid]
Then, starting index = mid + 1
Go to Step:1
Else if element <arr[mid]

Then,
last index = mid-1
Go to Step:2
Else:

{ means element == arr[mid] }
Print "Element Presented at position" + mid
Exit

PROGRAM :

```
def Binary_search(arr, start_index, last_index, element):

    while(start_index<=last_index):

                mid=int(start_index+last_index)/2)

                if(element>arr[mid]):
```

```
                start_index=mid+1
            elif(element<arr[mid]):
                last_index=mid-1
        elif(element==arr[mid]):
            return mid
        else
        return -1
arr=[]
n=input("enter no of elements:"))
for i in range(1,n+1):
    b=int(input("enter element"))
    arr.append(b)
print(arr)
    element=int(input("enter element to be searched"))
 start_index=0
 last_index=len(arr)-1
 found = Binary_search(arr, start_index, last_index, element)
 if (found = = -1):
print ("element not present in array")
else
    print("element is present at index", found)
```

OUTPUT 1:

Enter number of elements:8

Enter element:11

Enter element:33

Enter element:44

Enter element:56

Enter element:63

Enter element:77

Enter element:88
Enter element:90
[11, 33, 44, 56, 63, 77, 88, 90]
Enter the element to be searched
63 element is present at index 4

OUTPUT 2:

Enter number of elements:7
Enter element:11
Enter element:15
Enter element:20
Enter element:25
Enter element:30
Enter element:40
Enter element:50
[11, 15, 20, 25, 30, 40, 50] Enter the element to be searched 22
element not present in array

RESULT:

Thus the program to perform Binary Search is executed and the output is obtained.

Ex.No.4a **SELECTION SORT**

AIM:

To study and Implement Selection sort using python.

PRE LAB DISCUSSION

The provided Python code demonstrates the Selection Sort algorithm. Selection Sort has a time complexity of O(n^2). In each iteration, the code finds the minimum element's index in the unsorted portion of the array and swaps it with the current index's element. This gradually sorts the array from left to right. The example initializes an array, applies the selectionSort function to sort it, and then prints the sorted array in ascending order. The sorted array is obtained by repeatedly finding the smallest element in the unsorted portion and placing it in its correct position, resulting in an ordered array

ALGORITHM:

1. Start
2. Get the length of the array.
3. length = len(array) → 6
4. First, we set the first element as minimum element.
5. Now compare the minimum with the second element. If the second element is smaller than the first, we assign it as a minimum.
6. After each iteration, minimum element is swapped in front of the unsorted array.
7. The second to third steps are repeated until we get the sorted array.
8. Stop

PROGRAM:

```
# Selection sort in Python
# time complexity O(n*n)
#sorting by finding min_index
def selectionSort(array, size):

  for ind in range(size):
    min_index = ind
```

```
        for j in range(ind + 1, size):
            # select the minimum element in every iteration
            if array[j] < array[min_index]:
                min_index = j
        # swapping the elements to sort the array
        (array[ind], array[min_index]) = (array[min_index], array[ind])

arr = [-2, 45, 0, 11, -9,88,-97,-202,747]
size = len(arr)
selectionSort(arr, size)
print('The array after sorting in Ascending Order by selection sort is:')
print(arr)
```

OUTPUT

```
The array after sorting in Ascending Order by selection sort is:

[-202, -97, -9, -2, 0, 11, 45, 88, 747]
```

RESULT:

Thus the python program is implemented by selection sort to sort the given array.

Ex.No.4b **INSERTION SORT**

AIM:

To perform sorting using Insertion sort

PRE LAB DISCUSSION

The insertion Sort function takes an array arr as input. It first calculates the length of the array (n). If the length is 0 or 1, the function returns immediately as an array with 0 or 1 element is considered already sorted.

For arrays with more than one element, the function proceeds to iterate over the array starting from the second element. It takes the current element (referred to as the "key") and compares it with the elements in the sorted portion of the array that precede it. If the key is smaller than an element in the sorted portion, the function shifts that element to the right, creating space for the key.

.

ALGORITHM:

1. Start
2. We start with second element of the array as first element in the array is assumed to be sorted.
3. Compare second element with the first element and check if the second element is smaller then swap them.
4. Move to the third element and compare it with the second element, then the first element and swap as necessary to put it in the correct position among the first three elements.
5. Continue this process, comparing each element with the ones before it and swapping as needed to place it in the correct position among the sorted elements.
6. Repeat until the entire array is sorted.
7. Stop

PROGRAM:

```
def insertionSort(arr):
    n = len(arr)  # Get the length of the array
        if n <= 1:
        return  # If the array has 0 or 1 element, it is already sorted, so return
    for i in range(1, n):  # Iterate over the array starting from the second element
        key = arr[i]  # Store the current element as the key to be inserted in the right position
        j = i-1
        while j >= 0 and key < arr[j]:  # Move elements greater than key one position ahead
            arr[j+1] = arr[j]  # Shift elements to the right
            j -= 1
        arr[j+1] = key # Insert the key in the correct position
  # Sorting the array [12, 11, 13, 5, 6] using insertionSort
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print('The array after sorting in Ascending Order by insertion sort is:')
print(arr)
```

OUTPUT

```
The array after sorting in Ascending Order by insertion sort is:
[5, 6, 11, 12, 13]
```

RESULT :

Thus the python program to perform sorting using insertion sort technique is executed successfully.

EX. NO: 5a **MERGE SORT**

AIM

To perform sorting of the given data items using merge sort

PRE LAB DISCUSSION

Merge sort is a sorting algorithm that follows the **divide-and-conquer** approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

In simple terms, we can say that the process of **merge sort** is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

ALGORITHM:

1. Start
2. Check if the left index is less than the right index.
3. Calculate the midpoint of the array if the low index is less than the high index.
4. Call the mergesort function on the left and right halves of the array.
5. Merge the two sorted halves using the merge function.
6. Merge function creates two different arrays and copies the left and right halves into these arrays.
7. Iteration and comparison of both arrays are done.
8. After merging, the arrays are sorted in ascending order.
9. Stop

PROGRAM:

```
# Python program for implementation of MergeSort
# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)
    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    # Merge the temp arrays back into arr[l..r]
    i = 0     # Initial index of first subarray
    j = 0     # Initial index of second subarray
    k = l     # Initial index of merged subarray
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    # Copy the remaining elements of L[], if there
    # are any
    while i < n1:
        arr[k] = L[i]
```

```
        i += 1
        k += 1
    # Copy the remaining elements of R[], if there
    # are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr, l, r):
    if l < r:
        # Same as (l+r)//2, but avoids overflow for
        # large l and h
        m = l+(r-l)//2
        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)
# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is")
for i in range(n):
    print("%d" % arr[i],end=" ")
mergeSort(arr, 0, n-1)
print("\n\nSorted array is")
for i in range(n):
    print("%d" % arr[i],end=" ")
```

OUTPUT:

Given array is 12 11 13 5 6 7

Sorted array is 5 6 7 11 12 13

RESULT

Thus the python program to perform sorting using merge sort technique is executed successfully.

EX NO : 5b **QUICK SORT**

AIM:

To sort the given list of data items using quick sort techniques.

PRE LAB DISCUSSION

One of the most effective sorting algorithms is Quicksort, which is based on the divide-and-conquer strategy. Quicksort makes some average memories complexity of O(n log n) and is generally utilized practically speaking.

ALGORITHM:

Inputs:

A: an array of n elements

lo: the index of the first element of the sub-array to be sorted

hi: the index of the last element of the sub-array to be sorted

1. If lo is less than hi, then do the following:
   - Call partition(A, lo, hi) and store the index of the pivot element in p.
   - Recursively call quicksort(A, lo, p-1).
   - Recursively call quicksort(A, p+1, hi).

Partition Algorithm:

1. Let pivot be the last element of the sub-array A[lo..hi].
2. Let i be the index of the first element of the sub-array.
3. For each j from lo to hi-1, do the following:
1. If A[j] <= pivot, then do the following:
1. Increment i.
2. Swap A[i] with A[j].
4. Swap A[i+1] with A[hi].
5. Return i+1.

PROGRAM:

```
# Python program for Quicksort
def quicksort(arr, lo, hi):
    """
    Sorts the given array in ascending order using the Quicksort algorithm.
```

```
    Parameters:
    arr (list): The array to be sorted
    lo (int): The index of the first element in the sub-array to be sorted
    hi (int): The index of the last element in the sub-array to be sorted
    """
    if lo < hi:
        # Partition the array and get the index of the pivot element
        p = partition(arr, lo, hi)

        # Recursively sort the left and right partitions
        quicksort(arr, lo, p-1)
        quicksort(arr, p+1, hi)
def partition(arr, lo, hi):
    """
    Partitions the sub-array by selecting the last element as the pivot,
    and rearranging the array so that all elements to the left of the pivot
    are less than or equal to the pivot, and all elements to the right of the
    pivot are greater than the pivot.

    Parameters:
    arr (list): The array to be partitioned
    lo (int): The index of the first element in the sub-array to be partitioned
    hi (int): The index of the last element in the sub-array to be partitioned

    Returns:
    int: The index of the pivot element after partitioning
    """
    # Select the last element as the pivot
    pivot = arr[hi]
    i = lo - 1

    # Loop through the sub-array and partition it
```

```
    for j in range(lo, hi):
        if arr[j] <= pivot:
            # Move the element to the left partition
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    # Move the pivot element to its final position in the array
    arr[i+1], arr[hi] = arr[hi], arr[i+1]
    # Return the index of the pivot element
    return i+1
# Example usage
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quicksort(arr, 0, n-1)
print("The given array before sorting is : [10, 7, 8, 9, 1, 5] ")
print("Sorted array by quick sort is:", arr)
```

OUTPUT

```
The given array before sorting is : [10, 7, 8, 9, 1, 5]
Sorted array by quick sort is: [1, 5, 7, 8, 9, 10]
```

RESULT :

Thus the python program for sorting the given data items using quick sort is executed successfully.

EX NO: 6a IMPLEMENTING APPLICATIONS USING LISTS

Aim:

To write a python program to create, slice, change, delete and index elements using List.

PRE LAB DISCUSSION

In Python, list slicing is a common practice and it is the most used technique for programmers to solve efficient problems. Consider a Python list, in order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:). With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

Python List Slicing Syntax

The format for list slicing is of Python List Slicing is as follows:

Lst[Initial : End : IndexJump]

If *Lst* is a list, then the above expression returns the portion of the list from index *Initial* to index *End*, at a step size *IndexJump*.

ALGORITHM :

Step 1: Create the List.

Step 2: Indexing the List using the index operator [ ].

Step3: Silicing an element from the List

Step4: Step 4: Changing an element from the List.

Step 5: Appending the List.

Step 6: Removing an element from the List.

Step 7:Deleting an element from the List.

PROGRAM:

```
print("list is created in the name:list")

list=['p','e','r','m','i','t']

print('list created",list)

print("list indexing",list[0])

print("list negative indexing",list[-1])

print("list slicing",list[1:4])

list=['p','e','r','m','i','t']
```

```
print("Given list",list)

 list[0]=2

print("List Changing",list)

list[1:4]=[1,2,3]

print("List Changing",list)

list =
['p','e','r','m','i','t']

print("Given list",list)

list[0]=2

print("List Changing",list)

list[1:4]=[1,2,3]

print("List Changing",list)

list =
['p','e','r','m','i','t']

print("Given list",list)


list.append(['add','sub'])

print("List appending",list)

list  = ['p','e','r','m','i','t'  ]

print("Given list",list)

list.remove('p')

print("List Removing",list)

list = ['p','e','r','m','i','t']

print("Given list",list)

list[2:5] = []

print("List Delete",list)
```

OUTPUT :

List is created in the name: list

List Created ['p', 'e', 'r', 'm', 'i', 't']

List indexing p

List negative indexing t

List slicing ['e', 'r', 'm']

Given list ['p', 'e', 'r', 'm', 'i', 't']

List Changing [2, 'e', 'r', 'm', 'i', 't']

List Changing [2, 1, 2, 3, 'i', 't']

Given list ['p', 'e', 'r', 'm', 'i', 't']

List appending ['p', 'e', 'r', 'm', 'i', 't', ['add', 'sub']]

Given list ['p', 'e', 'r', 'm', 'i', 't']

List Removing ['e', 'r', 'm', 'i', 't']

Given list[‘p’,’e’,’r’,’m’,’i’,’t’]

List Delete[‘p’,’e’,’t’]

RESULT:

Thus program to create, slice, change, delete and index elements using list is executed and the output is obtained**.**

Ex. No: 6b **IMPLEMENTING APPLICATIONS USING TUPLE**

AIM:

To write a python program to create, slice, delete and index elements using Tuple.

PRE LAB DISCUSSION

Tuples are an immutable data structure in Python, meaning their elements cannot be changed once they are created. They are commonly used when you need to group multiple related values together in a compact and efficient way.

Characteristics of Tuples

Immutable: Once created, the elements in a tuple cannot be modified.
Ordered: Tuples maintain the order of elements.
Allows Duplicates: Elements in a tuple can be repeated.
Heterogeneous: Can store elements of different data types.

ALGORITHM:

1: start

2: create a tuple with empty, having integers,objects in different data types and nested tuples

3: slicing the tuple with : operator

4. Deleting the tuple with del method

5. Indexing the elements of tuple

6. Stop.

PROGRAM1:

 # Python program to show how to create a tuple

```
# Creating an empty tuple
empty_tuple = ()
print("Empty tuple: ", empty_tuple)


# Creating tuple having integers
int_tuple = (4, 6, 8, 10, 12, 14)
print("Tuple with integers: ", int_tuple)


# Creating a tuple having objects of different data types
mixed_tuple = (4, "Python", 9.3)
print("Tuple with different data types: ", mixed_tuple)

# Creating a nested tuple
nested_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6))
print("A nested tuple: ", nested_tuple)
```

OUTPUT:

Empty tuple: ()

Tuple with integers: (4, 6, 8, 10, 12, 14)

Tuple with different data types: (4, 'Python', 9.3)

A nested tuple: ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))

PROGRAM2:

```
# Python program to show how slicing works in Python tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Using slicing to access elements of the tuple
print("Elements between indices 1 and 3: ", tuple_[1:3])
# Using negative indexing in slicing
print("Elements between indices 0 and -4: ", tuple_[:-4])
# Printing the entire tuple by using the default start and end values.
print("Entire tuple: ", tuple_[:])
```

OUTPUT:

Elements between indices 1 and 3: ('Tuple', 'Ordered')

Elements between indices 0 and -4: ('Python', 'Tuple')

Entire tuple: ('Python', 'Tuple', 'Ordered', 'Immutable', 'Collection', 'Objects')

PROGRAM3:

```
# Python program to show how to delete elements of a Python tuple
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Deleting a particular element of the tuple
try:
    del tuple_[3]
    print(tuple_)
except Exception as e:
    print(e)
# Deleting the variable from the global space of the program
del tuple_
# Trying accessing the tuple after deleting it
```

```
try:
    print(tuple_)
except Exception as e:
    print(e)
```

OUTPUT:

'tuple' object does not support item deletion

name 'tuple_' is not defined

PROGRAM4:

```
# Creating tuples
Tuple_data = (0, 1, 2, 3, 2, 3, 1, 3, 2)
# getting the index of 3
res = Tuple_data.index(3)
print('First occurrence of 1 is', res)
# getting the index of 3 after 4th
# index
res = Tuple_data.index(3, 4)
print('First occurrence of 1 after 4th index is:', res)
```

OUTPUT:

First occurrence of 1 is 2

First occurrence of 1 after 4th index is: 6

RESULT:

Thus the program to illustrate the applications of tuple like create, slice, delete and index elements using Tuple has been executed successfully.

Ex. No: 7 IMPLEMENTING APPLICATIONS USING SETS & DICTIONARIES

AIM:

To write a python program to check if a Given String Is Hetero gram or Not using sets. &

To write a python program to add items to the inventory, Update items to the inventory and to display items using dictionaries

PRE LAB DISCUSSION:

A string is a heterogram if it has no alphabet that occurs more than once. For example, "NaukriLearning" is not a heterogram. However, "blackhorse" is a heterogram. You can follow the below steps to check if a given string is a heterogram or not.

- Separate all the alphabets from other any other characters (using list comprehension)
- Convert list of alphabets into set because set has unique elements (using set())
- Check if the length of the set is equal to number of alphabets
- If yes, then string is heterogram otherwise its not

Using the ord() function which returns the ASCII value. If the ASCII value of the alphabet is greater than or equal to that of 'a' and less than or equal to 'z' then add it to the list 'alphabets'

PATHFINDER, DUMBWAITER, and BLACKHORSE are example of Heterogram

ALGORITHM (for Hectogram Program):

1: start

2: create a tuple with empty, having integers,objects in different data types and nested tuples

3: slicing the tuple with : operator

4. Deleting the tuple with del method

5. Indexing the elements of tuple

6. Stop.

ALGORTITHM (for Inventory Program):

1. **Start**
2. Initialize an empty inventory (dictionary).
3. **Display Menu**
    1. Show the following options:
        1. Display Inventory
        2. Add/Update Item
        3. Remove Item
        4. Exit
4. **Choice Input**

    1. Take user input to choose an option (1 to 4).

5. **Process Based on Choice**
    1. **Choice 1: Display Inventory**
        1. If inventory is not empty, print all items and their quantities.
        2. Else, print "Inventory is empty."
    2. **Choice 2: Add/Update Item**
        1. Take the item name as input.
        2. Take the quantity of the item as input (ensure it's a positive integer).
        3. If the item exists, add the input quantity to the existing quantity.
        4. If not, add the item with the provided quantity.
        5. Print the updated item and its quantity.
    3. **Choice 3: Remove Item**
        1. Take the item name to remove as input.
        2. If the item exists, remove it and print that it was removed.
        3. If not, print "Item not found."
    4. **Choice 4: Exit**
        1. Exit the program.
6. **Repeat**
    1. Continue displaying the menu until the user selects "Exit."
7. **End**

PROGRAM (For Hectogram)

```
#check if string is heterogram or not

#sample string

str1 = "Naurkrilearning"

str2 = "blackhorse"

def check_heterogram(input):

    # separate out list of all alphabets

    list_of_alphabets = [ alph for alph in input if ( ord(alph) >= ord('a') and ord(alph) <= ord('z') )]


    # convert into set and compare lengths

    if len(set(list_of_alphabets))==len(list_of_alphabets):
```

```
        print ("Yes, the string '", input, "'is heterogram")
    else:
        print ("No, the string'", input, "'is not heterogram")
check_heterogram(str1)
check_heterogram(str2)
```

OUTPUT:

```
No, the string' Naurkrilearning 'is not heterogram
Yes, the string ' blackhorse 'is heterogram
```

No, the string ‘ Naurkrilearning ‘ is not heterogram

Yes, the string ‘ Blackhorse ’ is heterogram

Program (For Inventory):

```
inventory = {}

def display_inventory():
    if inventory:
        for item, quantity in inventory.items():
            print(f"{item}: {quantity}")
    else:
        print("Inventory is empty.")

def add_or_update_item():
    item = input("Enter item name: ")
    quantity = int(input(f"Enter quantity for {item}: "))
    inventory[item] = inventory.get(item, 0) + quantity
    print(f"{item} updated with {inventory[item]} in stock.")

def remove_item():
    item = input("Enter item to remove: ")
    if item in inventory:
        del inventory[item]
        print(f"{item} removed.")
    else:
```

```
        print(f"{item} not found.")

def main():
    while True:
        print("\n1. Display Inventory\n2. Add/Update Item\n3. Remove Item\n4. Exit")
        choice = input("Choose an option: ")

        if choice == "1":
            display_inventory()
        elif choice == "2":
            add_or_update_item()
        elif choice == "3":
            remove_item()
        elif choice == "4":
            break
        else:
            print("Invalid choice.")

if _name___== "_main_":
    main()
```

OUTPUT:

```
1. Display Inventory
2. Add/Update Item
3. Remove Item
4. Exit
Choose an option: 1
Inventory is empty.

1. Display Inventory
2. Add/Update Item
3. Remove Item
4. Exit
Choose an option: 1
Inventory is empty.

1. Display Inventory
2. Add/Update Item
3. Remove Item
```

4. Exit
Choose an option: 2
Enter item name: Smartphone
Enter quantity for Smartphone : 40
Smartphone updated with 40 in stock.

1. Display Inventory
2. Add/Update Item
3. Remove Item
4. Exit
Choose an option: 1
Smartphone : 40

RESULT:

Thus the program to illustrate the applications of set has been executed successfully & thus the program to illustrate the applications of dictionaries has been executed successfully

Ex. No: 8 **IMPLEMENTING PROGRAMS USING FUNCTIONS**

AIM:

To write a python program to implement program using functions

PRELAB DISCUSSION

Functions are one of the core building blocks of programming. They allow you to break down complex problems into smaller, reusable, and manageable parts, improving code readability, maintainability, and scalability.

ALGORITHM:

Step1:Start
Step2:Define the function
Step3:Check the given choice using if
Step4:If the choice matches then given arithmetic expression will be evaluated
Step5:If it not matches break the condition
Step5:End

PROGRAM/SOURCE CODE:

```
def add(x, y):
   return x + y
def subtract(x, y):
   return x - y
def multiply(x, y):
   return x * y
def divide(x, y):
   return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
   choice = input("Enter choice(1/2/3/4): ")
   if choice in ('1', '2', '3', '4'):
      try:
         num1 = float(input("Enter first number: "))
         num2 = float(input("Enter second number: "))
      except ValueError:
         print("Invalid input. Please enter a number.")
         continue
      if choice == '1':
         print(num1, "+", num2, "=", add(num1, num2))
      elif choice == '2':
         print(num1, "-", num2, "=", subtract(num1, num2))
      elif choice == '3':
         print(num1, "*", num2, "=", multiply(num1, num2))
      elif choice == '4':
```

```
            print(num1, "/", num2, "=", divide(num1, num2))
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
    else:
        print("Invalid Input")
```

OUTPUT:

```
Select operation.
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 1
Enter first number: 5
Enter second number: 6
5.0 + 6.0 = 11.0
Let's do next calculation? (yes/no): yes
Enter choice(1/2/3/4): 3
            Enter first number: 7
            Enter second number: 7
7.0 * 7.0 = 49.0
Let's do next calculation? (yes/no):
Enter choice(1/2/3/4): 4
Enter first number: 4
Enter second number: 4
4.0 / 4.0 = 1.0

Let's do next calculation? (yes/no):
Enter choice(1/2/3/4): no
Invalid
```

RESULT:

Thus the program is executed for design the calculator to perform arithmetic operations using functions and the output is obtained.

Ex. No: 9 IMPLEMENTING PROGRAMS USING STRINGS

AIM:

To implement Python program using Strings to count Vowels and Consonants

PRELAB DISCUSSION

1. Understanding the Problem

Given an input string, the goal is to:
Identify the vowels (`a, e, i, o, u`).
Identify the consonants (all alphabets that are not vowels).
Count and display the number of vowels and consonants in the string.

String Manipulation

Strings in Python are sequences of characters.
They are immutable, but you can process them character by character.

Character Checking

Use the `in` keyword to check if a character belongs to a group (e.g., vowels).
Use the `.isalpha()` method to ensure the character is a letter.

Iteration

Iterate over each character in the string using a `for` loop.

Case-Insensitive Comparison

Convert the string to lowercase using `.lower()` to ensure consistency.

ALGORITHM:

Step 1: **Initialize Counters**:

- Start with two counters, `v_count` and `c_count`, both set to zero. These will keep track of the number of vowels and consonants.

Step 2: **Define Vowels**:

- Create a string of all vowels (both uppercase and lowercase) for easy checking. For example, `vowels = "aeiouAEIOU"`.

Step 3: **Iterate Through Each Character**:

- Loop through each character in the input string.

Step 4: **Check if Character is Alphabetic**:

- For each character, first check if it's a letter using `isalpha()` to ignore spaces, punctuation, and other non-alphabetic characters.

Step 5: **Determine if Vowel or Consonant**:

- If the character is a letter, check if it's in the `vowels` string:
  - If it is, increment the `v_count`.
  - If it is not, it's a consonant, so increment the `c_count`.

Step 6: **Return or Print the Results**:

- Once the loop completes, print or return the values of `v_count` and `c_count`.

PROGRAM/SOURCE CODE:

```
def count_vowels_consonants(text):
    vowels = "aeiouAEIOU"
    v_count = 0
    c_count = 0
    for char in text:
        if char.isalpha(): # Check if the character is a letter
            if char in vowels:
                v_count += 1
            else:
                c_count += 1
    return v_count, c_count

# Example usage
user_input = input("Enter a sentence: ")
vowels, consonants = count_vowels_consonants(user_input)
print(f"Vowels: {vowels}, Consonants: {consonants}")
```

OUTPUT:

Enter a sentence: HELLO WORLD

Vowels: 3, Consonants: 7

RESULT:

Thus the program is executed for implement program using Strings to count Vowels and Consonants.

Ex. No: 10 **IMPLEMENTING PROGRAMS USING WRITTEN MODULES AND PYTHON STANDARD LIBRARIES (PANDAS,NUMPY, MATPLOTLIB, SCIPY)**

AIM:

To Implement python programusing written modules and Python Standard Libraries (pandas,numpy, Matplotlib, scipy)

PRE LAB DISCUSSION

- Load data from a CSV file using `pandas`.
- Clean and manipulate data using `numpy`.
- Visualize the data using `matplotlib`.
- Perform statistical analysis using `scipy`.

ALGORITHM

Step 1: **Define the Modules**:

- Create separate modules for loading data, processing data, visualizing data, and performing statistical analysis.

Step 2: **Module 1: Data Loading (`data_loader.py`)**:

- Define a function to load data from a CSV file using `pandas`.
- If the file is not found, print an error message.

Step 3: **Module 2: Data Processing (`data_processor.py`)**:

- Define functions to:
  - Calculate average scores for each subject using `numpy`.
  - Retrieve scores for a specific student.

Step 4: **Module 3: Data Visualization (`data_visualizer.py`)**:

- Define a function to plot a bar chart of the average scores by subject using `matplotlib`.

Step 5: **Module 4: Statistical Analysis (`data_analyzer.py`)**:

- Define a function to perform a statistical t-test between scores of two subjects using `scipy`.
- The function should return the t-statistic and p-value.

Step 6: **Main Program (`main_program.py`)**:

- Import the modules created in Steps 2–5.
- Load the data using the `data_loader` module.
- If data loading is successful:
  - Calculate and print the average scores.
  - Visualize the average scores by calling the `plot_averages` function from `data_visualizer`.
  - Retrieve and print scores for a specific student.
  - Perform a t-test between scores of two subjects and print the results.

Prepare the Data (Sample CSV)

Create a CSV file named `student_scores.csv` for the example.

student_scores.csv
Name,Math,Science,English
Alice,88,92,85
Bob,76,85,80
Charlie,90,78,88

Daisy,65,70,60
Evan,95,89,94

Step 2: Create Modules for Different Tasks

Module 1: Data Loader (data_loader.py)
This module will handle loading the CSV file using `pandas`.

```
import pandas as pd

def load_data(file_path):
    """Load data from a CSV file into a pandas DataFrame."""
    try:
        data = pd.read_csv(file_path)
        return data
    except FileNotFoundError:
        print("File not found.")
        return None
```

Module 2: Data Processor (data_processor.py)
This module will handle data manipulation and calculation of averages using numpy.

```
import numpy as np

def calculate_averages(data):
    """Calculate average scores for each subject."""
    averages = {
        "Math": np.mean(data['Math']),
        "Science": np.mean(data['Science']),
        "English": np.mean(data['English'])
    }
    return averages

def get_student_scores(data, student_name):
    """Get scores for a specific student."""
    student = data[data['Name'] == student_name]
    if not student.empty:
        return student[['Math', 'Science', 'English']].values[0]
    else:
        print("Student not found.")
        return None
```

Module 3: Data Visualizer (data_visualizer.py)
This module will handle data visualization using matplotlib.

```
import matplotlib.pyplot as plt

def plot_averages(averages):
    """Plot a bar chart of average scores."""
```

```
    subjects = list(averages.keys())
    scores = list(averages.values())

    plt.bar(subjects, scores, color=['blue', 'green', 'red'])
    plt.xlabel('Subjects')
    plt.ylabel('Average Score')
    plt.title('Average Scores by Subject')
    plt.show()
```

Module 4: Statistical Analysis (data_analyzer.py)
This module will use scipy to perform statistical analysis (e.g., t-test between two subjects).

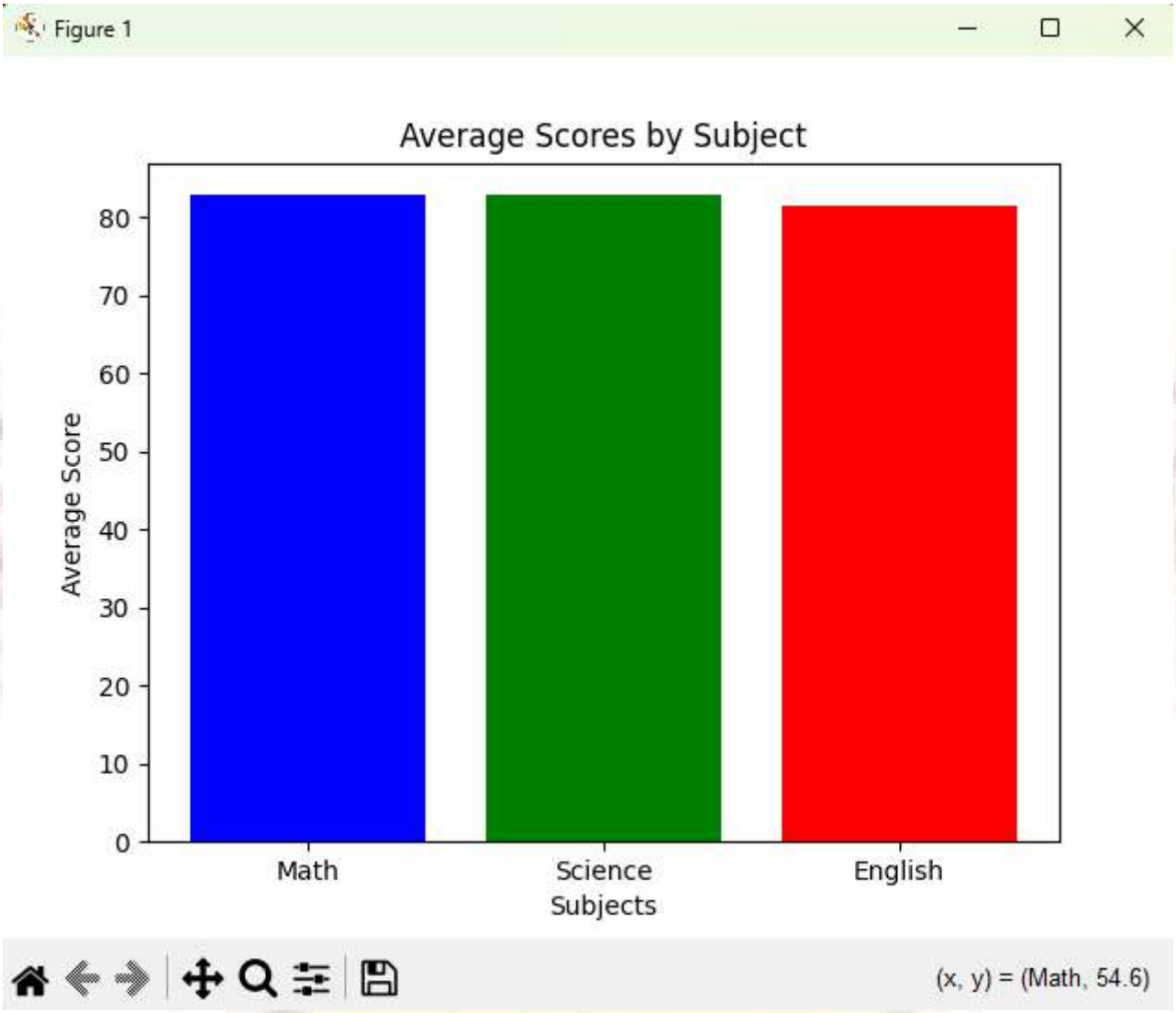
```
from scipy.stats import ttest_ind

def ttest_between_subjects(data, subject1, subject2):
    """Perform t-test between two subjects to check if their scores are significantly different."""
    scores1 = data[subject1]
    scores2 = data[subject2]
    t_stat, p_value = ttest_ind(scores1, scores2)
    return t_stat, p_value
```

Step 3: Main Program
Create the main program file that imports these modules and runs the analysis.

```
# main_program.py
import data_loader
import data_processor
import data_visualizer
import data_analyzer

# Load data
data = data_loader.load_data('student_scores.csv')
if data is not None:
    # Calculate averages
    averages = data_processor.calculate_averages(data)
    print("Average Scores:", averages)

    # Plot averages
    data_visualizer.plot_averages(averages)

    # Get scores for a specific student
    student_name = "Alice"
    scores = data_processor.get_student_scores(data, student_name)
    if scores is not None:
        print(f"Scores for {student_name}: Math={scores[0]}, Science={scores[1]}, English={scores[2]}")

    # Perform t-test between Math and Science scores
    t_stat, p_value = data_analyzer.ttest_between_subjects(data, 'Math', 'Science')
    print(f"T-test between Math and Science: t_stat={t_stat}, p_value={p_value}")
    if p_value < 0.05:
```

```
    print("There is a statistically significant difference between Math and Science scores.")
else:
    print("No statistically significant difference between Math and Science scores.")
```

OUTPUT



Average Scores: {'Math': np.float64(82.8), 'Science': np.float64(82.8), 'English': np.float64(81.4)}
Scores for Alice: Math=88, Science=92, English=85
T-test between Math and Science: t_stat=0.0, p_value=1.0
No statistically significant difference between Math and Science scores.

RESULT

Thus the python program is implemented and executed using all the written modules and Python Standard Libraries (pandas,numpy, Matplotlib, scipy)

Ex. No: 11 IMPLEMENTING REAL-TIME/TECHNICAL APPLICATIONS USING FILE HANDLING

AIM:

To write a python program to Implement time/technical applications using file handling

PRE LAB DISCUSSION:

Types of File Access

Python supports two main types of file access:

1. **Text Files**: Files containing plain text, e.g., `.txt`.
2. **Binary Files**: Files containing binary data, e.g., images, videos, or compiled programs.

Modes of File Handling

Python provides several modes for working with files:

Mode	Description
`r`	Open for reading (default mode).
`w`	Open for writing (overwrites file).
`a`	Open for appending (writes at end).
`rb`	Open for reading in binary mode.
`wb`	Open for writing in binary mode.

Basic Operations in File Handling

Opening a File

Files are opened using the `open()` function

Reading from a File

`read()`: Reads the entire file content.
`readline()`: Reads one line at a time.
`readlines()`: Reads all lines into a list.

Writing to a File

`write()`: Writes a single string to the file.
`writelines()`: Writes a list of strings to the file.

Closing a File

Always close the file after operations to free system resources:

Tips for Efficient File Handling

Always use the `with` statement to handle files.
Validate file paths and modes before performing operations.
Avoid hardcoding file paths; use dynamic input or configuration files.
Test programs with various file sizes and content types.

ALGORITHM

Step 1 : Initialize Log File:

- Define a file (e.g., `system_logs.txt`) to store the log entries.

- If the file does not already exist, create it or open it in append mode to avoid overwriting existing entries.

Step 2 : Define a Function to Write Log Entries:

- Create a function, `write_log(message)`, which:
  - Retrieves the current timestamp.
  - Formats the timestamp and the log message as a single line.
  - Appends the log entry to the log file.
- This function should add each new log entry at the end of the file without erasing previous logs.

Step 3 : Define a Function to Read Log Entries:

- Create a function, `read_logs()`, which:
  - Opens the log file in read mode.
  - Reads all lines in the file and stores them in a list or directly displays them.
  - Prints each log entry line by line to provide a full overview of past events.

Step 4 : Define a Function to Analyze Logs by Time Range:

- Create a function, `analyze_logs_by_date(start_date, end_date)`, which:
  - Accepts a start and end date as parameters.
  - Converts these date strings to date objects for comparison.
  - Opens the log file and reads each line.
  - For each log entry, extracts and converts the timestamp from the line.
  - Filters logs that fall within the specified date range.
  - Counts the number of entries within the date range and displays the filtered logs.

Step 5 : Main Program Execution:

- Use the defined functions in a main program section to:
  - Write several sample log entries to demonstrate logging.
  - Read and display all log entries.
  - Analyze logs within a specified date range and display the results.

PROGRAM

```
import os
from datetime import datetime

# Define log file name
LOG_FILE = "system_logs.txt"

def write_log(message):
    """Append a log entry with a timestamp to the log file."""
    with open(LOG_FILE, "a") as log_file:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        log_file.write(f"{timestamp} - {message}\n")
    print("Log written successfully.")

def read_logs():
    """Read and print all log entries from the log file."""
    if os.path.exists(LOG_FILE):
        with open(LOG_FILE, "r") as log_file:
            logs = log_file.readlines()
        for log in logs:
            print(log.strip())
    else:
        print("Log file does not exist.")

def analyze_logs_by_date(start_date=None, end_date=None):
    """Analyze logs within a specified date range."""
    if not os.path.exists(LOG_FILE):
        print("Log file does not exist.")
        return

    # Convert start and end date to datetime objects if provided
    start_date = datetime.strptime(start_date, "%Y-%m-%d") if start_date else None
    end_date = datetime.strptime(end_date, "%Y-%m-%d") if end_date else None

    with open(LOG_FILE, "r") as log_file:
        logs = log_file.readlines()

    # Filter and count logs within date range
    filtered_logs = []
    for log in logs:
        log_timestamp = datetime.strptime(log.split(" - ")[0], "%Y-%m-%d %H:%M:%S")
        if (not start_date or log_timestamp >= start_date) and (not end_date or log_timestamp <= end_date):
            filtered_logs.append(log.strip())

    # Print analysis results
    print(f"\nTotal Logs: {len(filtered_logs)}")
    if filtered_logs:
        print("\nFiltered Logs:")
        for log in filtered_logs:
            print(log)
    else:
```

```
        print("No logs found within the specified date range.")

if _name____== "___main___":
    # Writing logs
    write_log("System started.")
    write_log("User login successful.")
    write_log("File downloaded successfully.")
    write_log("Error encountered while accessing database.")

    # Reading all logs
    print("\n--- All Logs ---")
    read_logs()

    # Analyzing logs within a specific date range
    print("\n--- Logs from 2024-11-01 to 2024-11-04 ---")
    analyze_logs_by_date("2024-11-01", "2024-11-04")
```

OUTPUT

```
Log written successfully.
Log written successfully.
Log written successfully.
Log written successfully.

--- All Logs ---
2024-11-06 14:49:45 - System started.
2024-11-06 14:49:45 - User login successful.
2024-11-06 14:49:45 - File downloaded successfully.
2024-11-06 14:49:45 - Error encountered while accessing database.

--- Logs from 2024-11-01 to 2024-11-04 ---

Total Logs: 0
No logs found within the specified date range.
```

RESULT:

Thus python program to Implement time/technical applications using file handling is executed successfully.

Ex. No: 12 IMPLEMENTING REAL-TIME/TECHNICAL APPLICATIONS USING EXCEPTION HANDLING

AIM:

To write a python program to implement time/technical applications using exception handling

PRE LAB DISCUSSION:

- **`read_file(file_path)`**:
  - Attempts to read the specified file.
  - If the file is missing, `FileNotFoundError` is handled, displaying a message and returning `None` to indicate an error.
- **`process_data(data)`**:
  - Attempts to process each line of data by converting it to an integer.
  - If a line cannot be converted to an integer, a `ValueError` is raised. The function then handles the error, prints an error message, and returns `None` to stop further processing.
- **`write_file(file_path, data)`**:
  - Attempts to write processed data to the specified output file.
  - Adds a timestamp to track when the file was generated.
  - Catches any `IOError` during file writing, which could occur if there are permission issues.
  - A general `Exception` catch is added to handle any other unexpected errors.
- **`main()`**:
  - Coordinates the read, process, and write functions.
  - Checks the return values from each function to ensure any errors terminate the program gracefully.

ALGORITHM

Step1: **Define Input and Output Files**:

- Specify the file paths for the input and output files.

Step2: **Define a Function to Read Data from the File**:

- Try to open the input file in read mode.
- **Handle `FileNotFoundError`**:
  - If the file is missing, display an error message and return `None` to indicate the error.
- **Handle `IOError`**:
  - If there's an issue reading the file (e.g., permissions), display an error message and return `None`.
- If successful, read the file contents line by line and return the data.

Step3: **Define a Function to Process the Data**:

- Initialize an empty list to store the processed data.
- For each line in the input data:
  - Try to convert the line to an integer.
  - **Handle `ValueError`**:

        - If the conversion fails (e.g., non-integer data), print an error message indicating invalid data and return `None`.
    - If successful, perform calculations (e.g., squaring the integer).
    - Append the result to the processed data list.
- Return the processed data if all lines are processed successfully.

Step4: **Define a Function to Write Processed Data to the Output File**:

- Try to open the output file in write mode.
- **Handle `IOError`**:
    - If there's an issue writing to the file (e.g., lack of permissions), display an error message and stop the program.
- Write each entry from the processed data to the file, including a timestamp to indicate when the data was generated.
- If successful, display a message indicating data was written successfully.

Step5: **Main Program Logic**:

- Call the read function to load data from the input file.
- If `None` is returned, stop further processing and exit the program.
- Call the process function to process the data.
- If `None` is returned, stop further processing and exit the program.
- Call the write function to save the processed data to the output file.
- Display a final message indicating the end of the program.

Step6: **General Exception Handling**:

- Use a general `Exception` block to catch any unforeseen errors, display an error message, and safely exit the program.

PROGRAM:

```
from datetime import datetime

# Define input and output file names
INPUT_FILE = "data_input.txt"
OUTPUT_FILE = "data_output.txt"

def read_file(file_path):
    """Read lines from a file and return them as a
    list."""  try:
        with open(file_path, "r")
            as file: data =
            file.readlines()
        print("File read
        successfully.") return data
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
```

```
    except IOError:
        print("Error: Issue with reading the file.")
    return None

def process_data(data):
    """Convert each line to an integer and calculate the square."""
    processed_data = []
    try:
        for line in data:
            number = int(line.strip())  # Attempt to convert to integer
            squared = number ** 2  # Calculate square
            processed_data.append(f"{number} squared is {squared}")
        print("Data processed successfully.")
        return processed_data
    except ValueError as e:
        print(f"Error: Invalid data. Cannot convert to integer. {e}")
    return None

def write_file(file_path, data):
    """Write processed data to a file with a timestamp."""
    try:
        with open(file_path, "w") as file:
            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            file.write(f"Processed Data (generated on {timestamp}):\n")
            for line in data:
                file.write(line + "\n")
        print("Data written successfully.")
    except IOError:
        print("Error: Issue with writing to the file.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

def main():
    # Step 1: Read data from the input file
    data = read_file(INPUT_FILE)
    if data is None:
        print("Terminating program due to read error.")
        return

    # Step 2: Process the data
    processed_data = process_data(data)
    if processed_data is None:
        print("Terminating program due to processing error.")
        return

    # Step 3: Write processed data to the output file
    write_file(OUTPUT_FILE, processed_data)
```

```
if _name____== "_main_":
    main()
```

OUTPUT

input file `data_input.txt` contains the following valid data

4
7
15
-3

output in the console

File read successfully.
Data processed successfully.
Data written successfully.

the output file `data_output.txt` would be:

Processed Data (generated on 2024-11-06 14:46:12):
4 squared is 16
7 squared is 49
15 squared is 225
-3 squared is 9

RESULT:

Thus a python program to implement time/technical applications using file handling is executed successfully.

Ex. No: 13 **CREATING AND INSTANTIATING CLASSES**

AIM:

To write a python program for employee payroll processing using class and objects.

PRE LAB DISCUSSION:

Write a Python class Employee with attributes like emp_id, emp_name, emp_salary, and emp_department and methods like calculate_emp_salary, emp_assign_department, and print_employee_details.

Sample Employee Data:

"ADAMS", "E7876", 50000, "ACCOUNTING"

"JONES", "E7499", 45000, "RESEARCH"

"MARTIN", "E7900", 50000, "SALES"

"SMITH", "E7698", 55000, "OPERATIONS"

- Use 'assign_department' method to change the department of an employee.
- Use 'print_employee_details' method to print the details of an employee.
- Use 'calculate_emp_salary' method takes two arguments: salary and hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula:

overtime = hours_worked – 50

Overtime amount = (overtime * (salary / 50))

ALGORITHM:

Step1:Create a class employee

Step2:Define a function to calculate the salary

Step3:Define a function to assign the department of the employee

Step4:Define a function to print employee details

Step5:Create an object to access the data from the list

Step6:End

PROGRAM/SOURCE CODE :

```
class Employee:
    def __init__(self, name, emp_id, salary, department):
        self.name = name
        self.id = emp_id
        self.salary = salary
        self.department = department
    def calculate_salary(self, salary, hours_worked):
        overtime = 0
        if hours_worked > 50:
            overtime = hours_worked - 50
        self.salary = self.salary + (overtime * (self.salary / 50))
    def assign_department(self, emp_department):
        self.department = emp_department
    def print_employee_details(self):
        print("\nName: ", self.name)
        print("ID: ", self.id)
        print("Salary: ", self.salary)
        print("Department: ", self.department)
        print("------------------------")
employee1 = Employee("ADAMS", "E7876", 50000, "ACCOUNTING")
employee2 = Employee("JONES", "E7499", 45000, "RESEARCH")
employee3 = Employee("MARTIN", "E7900", 50000, "SALES")
employee4 = Employee("SMITH", "E7698", 55000, "OPERATIONS")
```

```
print("Original Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()
employee1.assign_department("OPERATIONS")
employee4.assign_department("SALES")
employee2.calculate_salary(45000, 52)
employee4.calculate_salary(45000, 60)
print("Updated Employee Details:")
employee1.print_employee_details()
employee2.print_employee_details()
employee3.print_employee_details()
employee4.print_employee_details()
```

OUTPUT:

```
Original Employee Details:
Name:  ADAMS
ID:  E7876
Salary:  50000
Department:  ACCOUNTING
-----------------------
Name: JONES
ID:  E7499
Salary:  45000
Department:  RESEARCH
```

Name: MARTIN

ID: E7900

Salary: 50000

Department: SALES

Name: SMITH

ID: E7698

Salary: 55000

Department: OPERATIONS

Updated Employee Details:

Name: ADAMS

ID: E7876

Salary: 50000

Department: OPERATIONS

Name: JONES

ID: E7499

Salary: 46800.0

Department: RESEARCH

Name: MARTIN

ID: E7900

Salary: 50000

Department: SALES

Name:

SMIT

H ID:

E7698

Salary:

66000.0

Departme

nt:

SALES

RESULT:

Thus the Program was executed to create employee payroll processing using classes and objects and the output is obtained

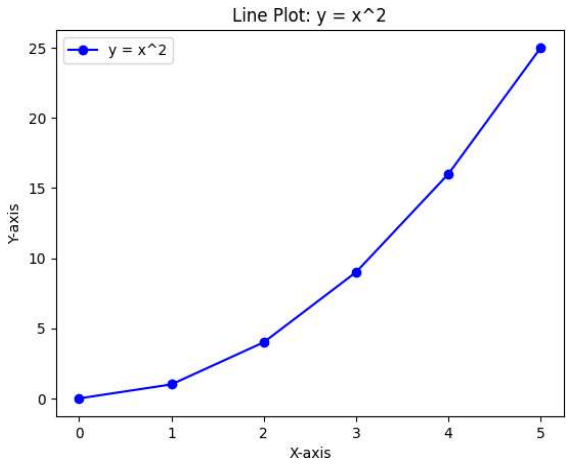
Ex. No: 14 **DATA VISUALIZATION TECHNIQUES**

AIM: To study and implement python program to implement various Data Visualizations using python Libraries

1. Matplotlib

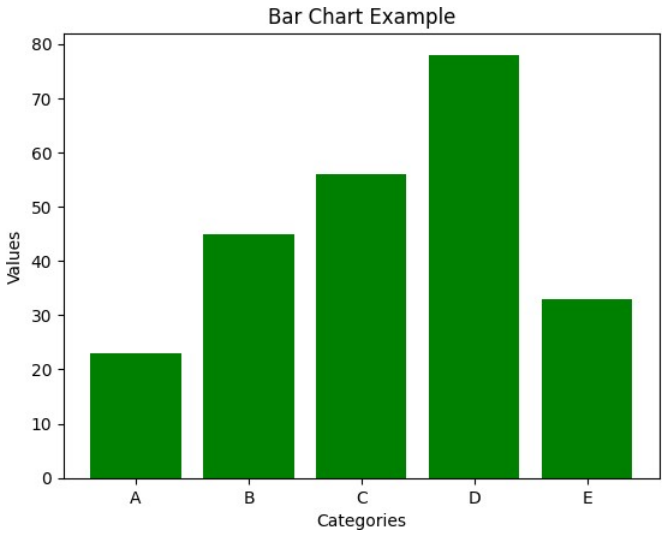
Matplotlib is a data visualization library and 2-D plotting library of Python It was initially released in 2003 and it is the most popular and widely-used plotting library in the Python community. It comes with an interactive environment across multiple platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, etc. It can be used to embed plots into applications using various GUI toolkits like Tkinter, GTK+, wxPython, Qt, etc. So you can use Matplotlib to create plots, bar charts, pie charts, histograms, scatterplots, error charts, power spectra, stemplots, and whatever other visualization charts you want! **The Pyplot module also provides a MATLAB-like interface that is just as versatile and useful as MATLAB while being free and open source.**

```
import matplotlib.pyplot as plt


# Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]
# Create a line plot
plt.plot(x, y, label="y = x^2", color="b", marker="o")
# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot: y = x^2")
plt.legend()
# Display the plot
plt.show()
```



Example 2:

```
import matplotlib.pyplot as plt
# Sample data
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 33]
# Create a bar chart
plt.bar(categories, values, color='red')
# Add labels and title
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Chart Example")
# Display the plot
plt.show()
```

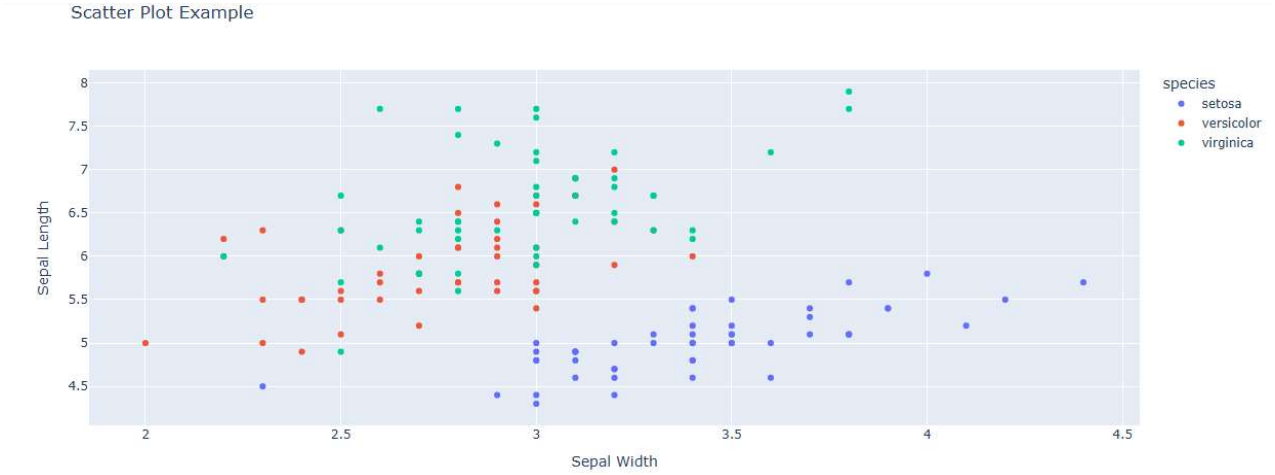


2. Plotly

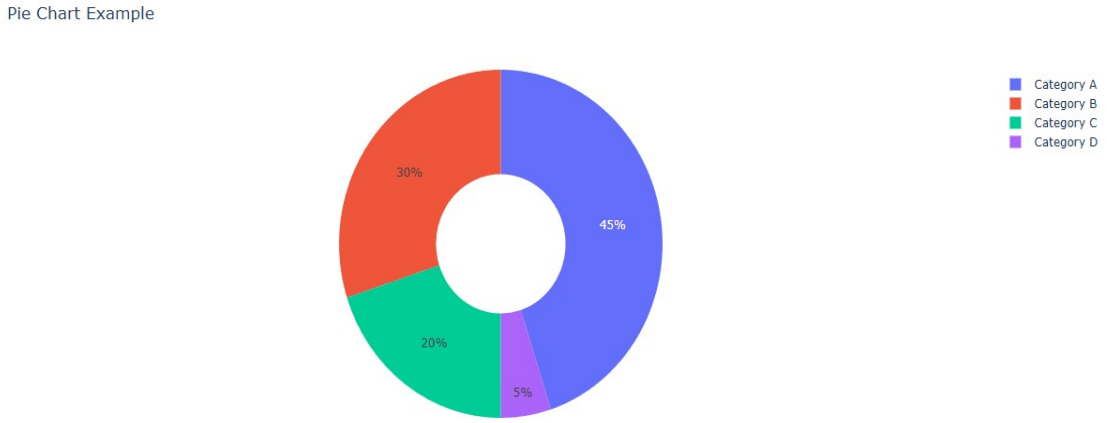
Plotly is a free open-source graphing library that can be used to form data visualizations. Plotly (plotly.py) is built on top of the Plotly JavaScript library (plotly.js) and can be used to create web-based data visualizations that can be displayed in Jupyter notebooks or web applications using Dash or saved as individual HTML files. Plotly provides more than 40 unique chart types like scatter plots, histograms, line charts, bar charts, pie charts, error bars, box plots, multiple axes, sparklines, dendrograms, 3-D charts, etc. Plotly also provides contour plots, which are not that common in other data visualization libraries. In addition to all this, Plotly can be used offline with no internet connection.

Example 4

```
import plotly.express as px
# Sample Data
df = px.data.iris()
# Create a scatter plot
fig = px.scatter(df, x='sepal_width', y='sepal_length', color='species',
                 title='Scatter Plot Example', labels={'sepal_width': 'Sepal Width', 'sepal_length': 'Sepal Length'})
fig.show()
```




```
import plotly.graph_objects as go
# Data
labels = ['Category A', 'Category B', 'Category C', 'Category D']
values = [450, 300, 200, 50]
# Create a pie chart
fig = go.Figure(go.Pie(labels=labels, values=values, hole=0.4))
fig.update_layout(title='Pie Chart Example')
fig.show()
```



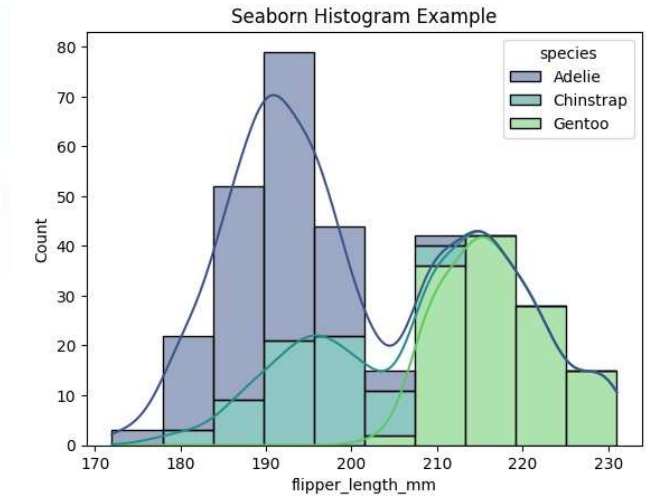
3. Seaborn

Seaborn is a Python data visualization library that is based on Matplotlib and closely integrated with the NumPy and pandas data structures. Seaborn has various dataset-oriented plotting functions that operate on data frames and arrays that have whole datasets within them. Then it internally performs the necessary statistical aggregation and mapping functions to create informative plots that the user

desires. It is a high-level interface for creating beautiful and informative statistical graphics that are integral to exploring and understanding data. The Seaborn data graphics can include bar charts, pie charts, histograms, scatterplots, error charts, etc. Seaborn also has various tools for choosing colour palettes that can reveal patterns in the data.

Example code for histogram

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load a sample dataset
data = sns.load_dataset('penguins')
# Create a histogram
sns.histplot(data=data, x='flipper_length_mm', hue='species', kde=True, multiple='stack', palette='viridis')
plt.title('Seaborn Histogram Example')
plt.show()
```

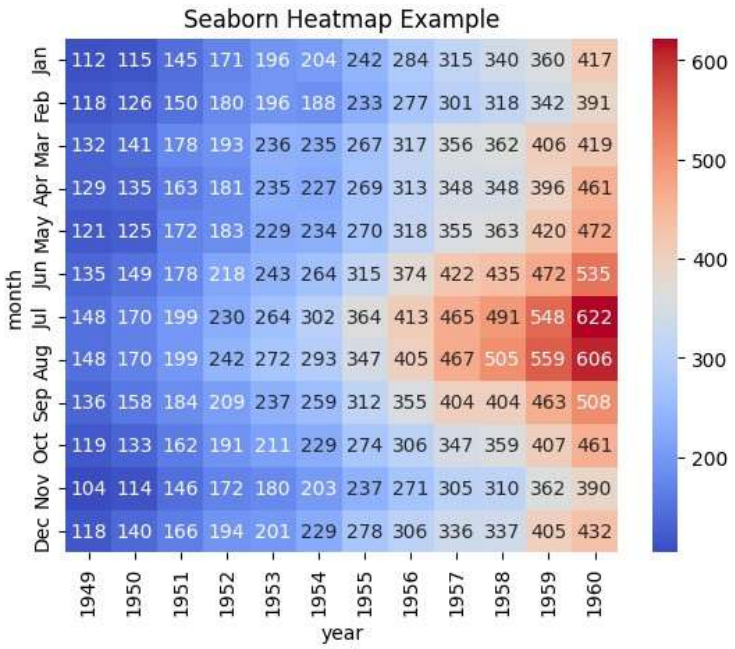


Example code for Heat map

```
import seaborn as sns
import matplotlib.pyplot as plt
# Create a pivot table
data = sns.load_dataset('flights')
```

```
# Use keyword arguments for index, columns, and values
pivot = data.pivot(index='month', columns='year', values='passengers')
# Create a heatmap
sns.heatmap(data=pivot, cmap='coolwarm', annot=True, fmt='d')
plt.title('Seaborn Heatmap Example')
plt.show()
```

output

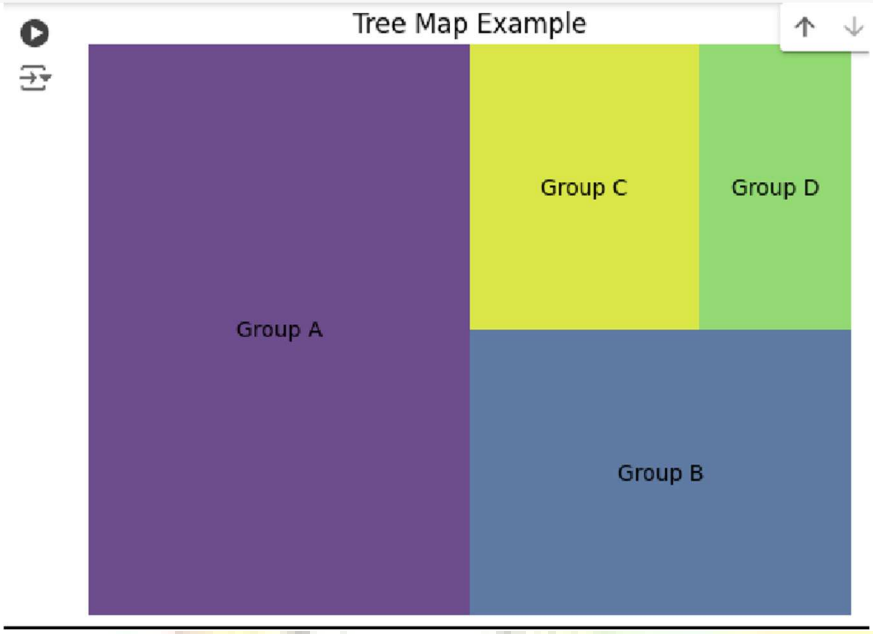


4. Squarify

Squarify is used for tree maps, which represent hierarchical data.

```
!pip install squarify
import squarify
import matplotlib.pyplot as plt

# Data
sizes = [50, 25, 15, 10]
labels = ['Group A', 'Group B', 'Group C', 'Group D']

# Create Tree Map
squarify.plot(sizes=sizes, label=labels, alpha=0.8)
```

```
plt.axis('off')
plt.title('Tree Map Example')
plt.show()
```



Result:

Thus the study and implementation of python program to display data visualizations using python libraries.

VIVA QUESTIONS

1. Basics of Python

1. What is Python, and what are its key features?
2. How is Python different from other programming languages like Java or C++?
3. Explain the concept of an interpreter in Python.
4. What are Python's main data types?
5. What is the difference between `is` and `==` in Python?

2. Python Syntax and Semantics

6. How do you define a variable in Python?
7. What are Python keywords, and how many are there in the current version?
8. Explain the significance of indentation in Python.
9. What are comments in Python, and how are they written?
10. Can Python have multiline comments? If yes, how?

3. Control Flow

11. What is the difference between `if`, `elif`, and `else` in Python?
12. How does a `while` loop differ from a `for` loop in Python?
13. What is the purpose of the `break` and `continue` statements?
14. How can you iterate over a range of numbers in Python?
15. Explain list comprehensions with an example.

4. Functions and Modules

16. What are functions, and why are they used in Python?
17. How do you define and call a function in Python?
18. What are default arguments and keyword arguments in Python functions?
19. How do you import a module in Python? Give an example.
20. What is the difference between `import` and `from ... import`?

5. File Handling

21. How do you open a file in Python?
22. Explain the difference between reading (`r`), writing (`w`), and appending (`a`) modes.
23. What is the `with` statement in file handling?
24. How can you read a file line by line in Python?
25. What happens if you try to read a file that doesn't exist?

6. Object-Oriented Programming (OOP)

26. What is a class in Python, and how is it different from an object?
27. What is the significance of the `self` keyword in Python classes?
28. How do you create an instance of a class in Python?

29. What are instance variables and class variables in Python?
30. Explain inheritance and polymorphism in Python.

7. Exception Handling

31. What are exceptions in Python?
32. How is exception handling implemented in Python?
33. What is the difference between `try-except` and `try-finally`?
34. What is the use of the `raise` keyword in Python?
35. How can you create custom exceptions in Python?

8. Python Collections

36. What is the difference between a list, tuple, set, and dictionary in Python?
37. How can you access elements in a tuple?
38. What is the significance of the `keys()` and `values()` methods in dictionaries?
39. How do you remove duplicate elements from a list in Python?
40. How is slicing implemented in Python lists?

9. Advanced Topics

41. What are decorators in Python?
42. How does Python manage memory?
43. Explain the difference between mutable and immutable objects in Python.
44. What is the difference between shallow copy and deep copy?
45. How does Python handle multi-threading?

10. Libraries and Frameworks

46. What are some commonly used Python libraries for data analysis?
47. Explain the difference between NumPy and Pandas.
48. How is Django different from Flask?
49. What is the purpose of Matplotlib in Python?
50. How can you install third-party libraries in Python?