Python Programs
Program 1

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.

Source Code:

```
import sys

print 'Number of arguments:', len(sys.argv), 'arguments.'
print 'Argument List:', str(sys.argv)
```

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']

Program2

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

Source code:

```
# 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]]
# result is 3x4
result = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]
if(len(X[0])==len(Y)):
fori in range(len(X)):#3
   # iterate through columns of Y
for j in range(len(Y[0])):#4
       # iterate through rows of Y
for k in range(len(Y)): #3
result[i][j] += X[i][k] * Y[k][j]
else:
print("Not Possible")

for r in result:
print(r)
```

Output

```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
```

Program-3

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

Source Code:

```
defgcd(x, y):
gcd=1

if x % y ==0:
return y

for kinrange(int(y /2),0,-1):
if x % k ==0and y % k ==0:
gcd= k
break
returngcd

print(gcd(12,17))
print(gcd(4,6))
```

Output:

```
1
2
```

Program-3

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

Program

```
fname=input("enter file name")
count=0            #count of a specific word
maxcount=0         #maximum among the count of each words
l=[]               #list to store the words with maximum count
with open(fname,'r') as f:
        contents=f.read()
        words=content.split()
fori in range(len(words)):
for j in range(len(words)):
if(words[i]==words[j]):       #finding count of each word
count+=1
else:
count=count
if(count==maxcount):          #comparing with maximum count
l.append(words[i])
elif(count>maxcount):         #if count greater than maxcount
l.clear()
l.append(words[i])
maxcount=count
else:
         l=l
count=0
print(l)
```

Output:

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

Hi, friends this program is found in text.
This program works perfectly

Print Output- [program]

Program-5

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.

$$\sqrt{a} = x$$

$$a = x^2$$

For,

$$f(x) = x^2 - a$$

$$f'(x) = 2 * x$$

$$\frac{f(x)}{f'(x)} = \frac{x^2 - a}{2 * x} = \frac{x - a/x}{2}$$

Since,

$$x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{x_n - a/x_n}{2}$$

$$x_{n+1} = \frac{x_n + a/x_n}{2}$$

Source Code:

```
defnewton_method(number, number_iters = 500):
    a = float(number) # number to get square root of
fori in range(number_iters): # iteration number
number = 0.5 * (number + a / number) # update
            # x_(n+1) = 0.5 * (x_n +a / x_n)
return number

printnewton_method(9)
# Output: 3
printnewton_method(2)
# Output: 1.41421356237
```

Program 6

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.

Program:

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

Output: Enter base: 7
Enter exponential value: 2
Result:49

Program-7

Object: To write a python program find the maximum of a list of numbers.

```
# list of numbers
list1 = [10, 20, 4, 45, 99]


# printing the maximum element
print("Largest element is:", max(list1))
```

Output- 99

Program 8

Object: Write a Python Program to perform Linear Search

Aim: To write a Python Program to perform Linear Search

Algorithm:

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

Program:

```
items = [5, 7, 10, 12, 15]
print("list of items is", items)
x = int(input("enter item to search:")
i = flag = 0
whilei<len(items):
        if items[i] == x:
                flag = 1
                        breaki = i + 1
        if flag == 1:
                print("item found at position:", i + 1)
        else:
                print("item not found")
```

Output:

```
(list of items is: [5, 7, 10, 12, 15] )
enter item to search: 7
(item found at position:, 2)
```

Program-9

Object: To write a python program Binary search.

Algorithm:

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. 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.
4. Else (x is smaller) recur for the left half.

```
# Python3 Program for recursive binary search.
 # Returns index of x in arr if present, else -1

defbinarySearch (arr, l, r, x):

   # Check base case
   ifr >=l:
      mid =l +(r -l) //2

      # If element is present at the middle itself
      ifarr[mid] ==x:
         returnmid

      # If element is smaller than mid, then it can only be present in left subarray
      elifarr[mid] > x:
         returnbinarySearch(arr, l, mid-1, x)

      # Else the element can only be present in right subarray
      else:
         returnbinarySearch(arr, mid +1, r, x)

   else:
      # Element is not present in the array
      return-1
```

```
# Driver Code
arr =[ 2, 3, 4, 10, 40]
x =10

# Function call
result =binarySearch(arr, 0, len(arr)-1, x)

ifresult !=-1:
   print("Element is present at index % d"%result)
else:
   print("Element is not present in array")
```

Output:

Element is present at index 3

Program-10

Object: To write a python program selection sort.

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.

```
# Python program for implementation of SelectionSort
importsys
A =[64, 25, 12, 22, 11]

# Traverse through all array elements
fori inrange(len(A)):
    # Find the minimum element in remaining unsorted array
   min_idx =i
   forj inrange(i+1, len(A)):
      ifA[min_idx] > A[j]:
         min_idx =j

   # Swap the found minimum element with the first element
   A[i], A[min_idx] =A[min_idx], A[i]

# Driver code to test above
print("Sorted array:", A)
```

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

Program-11

Object: To write a python program Insertion sort.

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

```
# Python program for implementation of Insertion Sort

# Function to do insertion sort
definsertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
         key = arr[i]
  # Move elements of arr[0..i-1], that are greater than key, to one position
ahead of their current position
        j = i-1
        while j >=0 and key <arr[j] :
               arr[j+1] = arr[j]
               j -= 1
        arr[j+1] = key

 # Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print ("Sorted array is:",arr)
```

Output:

Sorted array is: 5 6 11 12 13

Program-12

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] andarr[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)

The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.

```
# Python program for implementation of MergeSort
defmergeSort(arr):
    iflen(arr) >1:
        mid =len(arr)//2#Finding the mid of the array
        L =arr[:mid] # Dividing the array elements
        R =arr[mid:] # into 2 halves

        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half
```

```
        i =j =k =0

        # Copy data to temp arrays L[] and R[]
        whilei<len(L) andj <len(R):
            ifL[i] < R[j]:
                arr[k] =L[i]
                i+=1
            else:
                arr[k] =R[j]
                j+=1
            k+=1

        # Checking if any element was left
        whilei<len(L):
            arr[k] =L[i]
            i+=1
            k+=1

        whilej <len(R):
            arr[k] =R[j]
            j+=1
            k+=1

# Code to print the list
defprintList(arr):
    fori inrange(len(arr)):
        print(arr[i],end=" ")
    print()

# driver code to test the above code
if__name__ =='__main__':
    arr =[12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)
```

Output:
Given array is

12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13

Program-13

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)

Program:

```
n = int(input("Enter the upper limit: "))
print("Prime numbers are")
fornum in range(0,n + 1): # prime numbers are greater than 1
ifnum> 1:
fori in range(2,num):
if (num % i) == 0:
break
else:
print(num)
```

Sample Output:

Enter the upper limit: 20

Prime numbers are

2 35 7 11 13 17 19

Program 14

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

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

```
# Program


import sys, pygame
pygame.init()


size = width, height = 700, 300
speed = [1, 1]
background = 255, 255, 255


screen = pygame.display.set_mode(size)
pygame.display.set_caption("Bouncing ball")


ball = pygame.image.load("ball.jpg")
ballrect = ball.get_rect()
```

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

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

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

output: