Exercise 1. **Compute the GCD of two numbers.**

Aim: **To Compute the GCD of two number using Python.**

Algorithm:

- **Start**
- **Read num1, num2 to find the GCD**
- **If x>y**
    - **Smaller = y**
    - **Else**
    - **Smaller = x**
- **For i - smaller+1**
    - **If x%i==0 and y%i==0**
    - **Return gcd**
- **Call fun and Print gcd(num1,num2)**
- **Stop**

Program:

```
def GCD(x, y):
   if x > y:
      smaller = y
   else:
      smaller = x
   for i in range(1, smaller+1):
      if((x % i == 0) and (y % i == 0)):
         gcd = i
   return gcd
num1 = 20
num2 = 10
print("The GCD. of", num1,"and", num2,"is", GCD(num1, num2))
```

Output:

```
The GCD.of 20 and 10 is 10
```

Exercise 2 **Find the square root of a number (Newton‘s method)**

Aim: **To find the squareroot of a number using python.**

Algorithm:

```
• Start
• Read the input from n,a
• Approx = 0.5*n
• For i upto range a
    o Betterapprox = 0.5*(approx+n/approx)
    o Approx = betterapprox
• Return betterapprox
• Call the function and print newtonsqrt(n,a)
• Stop
```

Program:

```
def newtonSqrt(n, a):
    approx = 0.5 * n
    for i in range(a):
        betterapprox = 0.5 * (approx + n/approx)
        approx = betterapprox
    return betterapprox
print(newtonSqrt(10, 3))
print(newtonSqrt(10, 5))
print(newtonSqrt(10, 10))
```

Output:

```
3.162319422150883
3.162277660168379
3.162277660168379
```

Exercise 3. Exponentiation **(power of a number)**

Aim: **To find the exponentiation using python programming**

Algorithm:

```
• Start
• Read base value number in base
• Read exponent value number in exp
• If exp is equal to 1
    o Return base
• If exp is not equal to 1
    o Return (base*powerexp(base, exp-1)
• Call function Print the result
• Stop
```

Program:

```python
def powerexp(base,exp):

   if(exp==1):

       return(base)

   if(exp!=1):

       return(base*powerexp(base,exp-1))

base=int(input("Enter base Value: "))

exp=int(input("Enter exponential Value: "))

print("Result:",powerexp(base,exp))
```

Output:

```
Enter base Value: 5
Enter exponential Value: 3

Result: 125
```

Exercise 4: **Find the maximum of a list of numbers**

Aim: **To find the maximum of a list of numbers using python.**

Algorithm:

- **Start**
- **Read number of elements of the list**
- **Using loop until** n-1
    - **Read thr element user given in** b
- **Append all the elements in *a***
- **Repeat 4th step upto *n-1***
- **Sorting a**
- **Print the maximum of a list of number**
- **Stop**

Program:

```
a=[]

n=int(input("Enter number of elements:"))

for i in range(1,n+1):

    b=int(input("Enter element:"))

    a.append(b)

a.sort()

print("Maximum of a List of Number is:",a[n-1])
```

Output:

```
Enter number of elements:  5
Enter element: 5
Enter element: 8
Enter element: 2
Enter element: 1
Enter element: 8
Maximum of a List of Number is: 24
```

Exercise 5: **Linear search and Binary search**

Aim: **To find the value using linear search in python program.**

Algorithm:

- **Start**
- **Read n elements to list**
- **If I > n then go to step 7**
- **If A[i] = x then go to step 6**
- **Set I to I + 1**
- **Go to step 2**
- **Print elements x Found at index I And go to step 8**
- **Print element not found**
- **Stop**

Program:

```
def linearSearch(alist, item):

        pos = 0

        found = False


        while pos < len(alist) and not found:

            if alist[pos] == item:

                found = True

            else:

                pos = pos+1


        return found


linearlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]

print(linearSearch(linearlist, 3))

print(linearSearch(linearlist, 13))
```

Aim: **To find the value using binary search in python program.**

Algorithm:

- **Start**
- **Read the array elements**
- **Find the middle element in the sorted list**
- **Compare, the search element with middle element in the sorted list**
- **If both are matching print “Item Has been found”**
- **If the element also doesn’t match with the search element, then print “Items Not Found”**
- **Stop**

Program:

```
def binarySearch(alist, item):

        first = 0

        last = len(alist)-1

        found = False


        while first<=last and not found:

            midpoint = (first + last)//2

            if alist[midpoint] == item:

                found = True

            else:

                if item < alist[midpoint]:

                    last = midpoint-1

                else:

                    first = midpoint+1


        return found
```

```
blist = [0, 1, 2, 8, 13, 17, 19, 32, 42]

print(binarySearch(blist, -1))

print(binarySearch(blist, 13))
```

Exercise 6: **Selection sort, Insertion sort**

Aim: **To sort list of elements using selection sort.**

Algorithm:

- **Start**
- **Read upper Limit n**
- **Read n elements to the list**
- **For I = I to len(sample)**
    - **while(j <len(sample))**
    - **Repeat the steps until condition satisfied**
- **Call function Selsort() print the sorted elements.**

Program:

```
def selsort(sample):
 print("intial list:",sample)

 for i in range(len(sample)):
        print(sample)
        minIndex=i
        j = i + 1
        while(j < len(sample)):
                if(sample[j] < sample[minIndex]):
                        minIndex = j
                j+=1
        sample[i], sample[minIndex] = sample[minIndex], sample[i]
 print("sorted list",sample)

sample1 = [12,1,3,2,7,-100]
selsort(sample1)
```

Aim: **To sort list of elements using insertion sort.**

Algorithm:

```
• Start
• Read upper Limit n
• Read n elements to the list
• For I = I to len(sample)
    o while(j!=0 and sample[j] < sample[j-1])
    o Repeat the steps until condition satisfied

Call function insertsort () print the sorted elements
```

Program:

```
def insertsort(sample):

    print("intial sample:", sample)


    for i in range(1, len(sample)):

        print(sample)

        j=i

        while(j!=0 and sample[j] < sample[j-1]):

            sample[j-1], sample[j] = sample[j], sample[j-1]

            j-=1


    print("sorted list:",sample)


sample1 = [12,300,-90,-100-1000,1,4]

insertsort(sample1)
```

Exercise 7: **Merge sort**

Aim: **To sort list of elements using merge sort.**

Algorithm

- **Start**
- **Divide the arrays in left sub array & right sub array**
- **Conquer by recursively sorting the two sub arrays**
- **Combine the elements back in by merging the two sorted sub arrays**
- **Call the results and print the arrays**
- **Stop**

Program:

```
def merge(left,right):
    result = []
    i,j = 0, 0
    while i<len(left) and j<len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i+=1
        else:
            result.append(right[j])
            j+=1

    result += left[i:]
    result += right[j:]
    returnresult
defmergesort(lst):
    if(len(lst) <= 1):
        returnlst
    mid = int(len(lst)/2)
    left = mergesort(lst[:mid])
    right = mergesort(lst[mid:])
    return merge(left,right)
arr = [1,2,-1,0,9,65,7,3,4,1,2]
print(mergesort(arr))
```

Exercise 8: **First n prime numbers.**

Aim: **To write a program to find the prime number.**

Algorithm:

```
Start
Read p
Set q = 0
For I ->2 to p/2
If p % I == 0
Print Number is not prime
If q<= 0
Print Number is not prime
```

Program:

```
p=int(input("Enter number: "))
q=0
for i in range(2,p//2):
   if(p%i==0):
       q=q+1
if(q<=0):
   print("Number is prime")
else:
   print("Number isn't prime")
```

Output:

```
Enter Number: 8
Number isn’t  prime
Enter Number: 5
Number is prime
```

Exercise 9**. Multiply matrices**

Aim: **To Multiply Matrices using python.**

Algorithm:

- **Start**
- **Read matrix x and read matrix y**
- **Loop for each row in matrix X**
- **Loop for each columns in matrix Y**
- **Initialize output matrix Result to 0. This loop will run for each rows of matrix X.**
- **Multiply** X[i][k] * Y[k][j] and this value to result[i][j]
- **Print Matrix Result**
- **Stop**

Program:

```
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]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
     # iterate through rows of Y
     for k in range(len(Y)):
        result[i][j] += X[i][k] * Y[k][j]


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

Output:

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

Exercise 10: **Programs that take command line arguments(word count)**

Aim: **To find the word and lines in command line arguments.**

Algorithm:

- **Start**
- **Add arguments to find the words and lines**
- **Add file name as argument**
- **Parse the arguments to get the values**
- **Format and print the words**
- **Stop**

Program:

```
fname = input("Enter file name: ")

num_words = 0

with open(fname, 'r') as f:

    for line in f:

        words = line.split()

        num_words += len(words)

print("Number of words:")

print(num_words)
```

Output:

```
Enter file name: prem.txt
Number of words: 27
```

Exercise 11: **Find the most frequent words in a text read from a file.**

Aim: **To find the most frequent words in a text read from a file.**

Algorithm:

- **Start**
- **Read the filename**
- **Open the file**
- **Read each line from the file to count the lowers and words**
- **Split each line in to words and count them**
- **Print the word and counts**
- **Stop**

Program:

```
fr = open("prempaul.txt","r")

wordcount = {}

for word in fr.read().split():

    if word not in wordcount:

        wordcount[word] = 1

    else:

        wordcount[word] += 1

for k,v in wordcount.items():

    print(k, v)

fr.close()
```

Output:

```
To - 1
 Find - 1
 The - 1
 Most-1
 Frequent-1
 Words-1
 In-1
A-2
Text-1
Read-1
From-1
File-1
```