|  |  |
| --- | --- |
| **Ex. 4** | **EXPLORING LOOPS** |
| **Date:12/02/24** |  |

Aim:

To explore the use of loops in Python by writing programs for the following and

executing them:

1. Print the list of prime numbers between 1 and N.
2. Print the multiplication table up to M for a number N.
3. Print the following pattern for 2N-1 rows. 1

2 2

3 3 3

2 2

1

1. Find the greatest common divisor of 2 numbers obtained from the user.
2. Find the sum of the series 1, −(12), 23, −(34), 55, −(86), 137, … up to N terms.
3. Find the sum of the digits of a given integer, N.
4. Find the square root of an integGer, N, using Newton's method. Obtain N and the limit, L, from the user.

Algorithm:

(a)

Step 1: Prompt the user to input an integer n .

Step 2: Initialize an empty list to store prime numbers.

Step 3: Loop through numbers from 2 to n-1 :

- Initialize a counter for each number to track divisibility.

- Check divisibility of the number by iterating through possible divisors from 2 up to half the number.

- If the number is divisible by any divisor, increment the counter and break out of the loop.

- If the counter remains zero after checking all divisors, add the number to the list as a prime number.

Step 4: Print the list of prime numbers.

(b)

Step 1: Prompt the user to input integers n (number for which the table is to be printed) and m (number of multiples).

Step 2: Iterate from 1 to m:

- Print each multiplication of n with the current iterator i, displaying the result.

(C)

Step 1: Prompt the user for the number of rows n .

Step 2: Initialize a variable `space` equal to n to control leading spaces.

Step 3: For the upper half of the diamond (including the middle line):

- Print the necessary spaces to align the numbers centrally.

- Print numbers in ascending order from 1 up to the current row number, each repeated the same number of times as the row number.

- Decrease `space` by 1 after each row to reduce the number of spaces in the next line.

Step 4: Initialize another spacing variable `space1` for the lower half of the diamond.

Step 5: For the lower half of the diamond:

- Print the necessary spaces for alignment similar to the upper half.

- Print numbers in descending order from n-1 to 1, each number repeated as per its row number.

- Increase `space1` by 1 after each row to increase the number of spaces for alignment.

(d)

Step 1: Prompt the user to input two integers, a and b.

Step 2: Execute a while loop that continues as long as b is not zero:

- Inside the loop, set a to b and b to a modulo b, which repeatedly assigns b the remainder of a divided by b and a the old value of b.

Step 3: Once b becomes zero, print a as it now holds the GCD of the two numbers.

(e)

Step 1: Prompt the user to enter the number of terms, `n`, for the series.

Step 2: Initialize two variables `t1` and `t2` to represent the first two terms of the Fibonacci sequence, where `t1 = 0` and `t2 = 1`.

Step 3: Create an empty list `lst` and append `t2` to it to start the Fibonacci sequence.

Step 4: Generate the Fibonacci sequence up to `n` terms:

- Calculate the next term by adding `t1` and `t2`.

- Update `t1` to `t2` and `t2` to the new term (`c`).

- Append each new term to the list `lst`.

Step 5: Initialize `sum` to 0 and `num` to 0 to compute the series sum where each term is raised to an incrementally increasing power and alternately added or subtracted based on its index.

Step 6: Iterate over each Fibonacci number in `lst`:

- Use the index plus one as the exponent for `(-1)` to alternate the sign.

- Raise the Fibonacci number to the power of `num+1` (incrementing power).

- Accumulate these values in `sum`.

Step 7: After processing all terms, print the resulting `sum` of the series.

(f)

Step 1: Prompt the user to input an integer, `num`.

Step 2: Initialize `sum` to 0 to store the cumulative sum of the digits.

Step 3: Execute a while loop as long as `num` is not zero:

- Extract the last digit of `num` by taking `num % 10` and store it in `newnum`.

- Add `newnum` to `sum`.

- Remove the last digit from `num` by performing integer division `num // 10`.

Step 4: After exiting the loop, print the value of `sum`, which now contains the sum of all the digits of the input number.

(g)

Step 1: Define a function `newtons\_method\_sqrt` that takes two parameters: `N` (the number to find the square root of) and `L` (the tolerance level for approximation accuracy).

Step 2: Initialize `X` with the value of `N` to start the approximation.

Step 3: Enter a loop to repeatedly refine the approximation:

- Compute the next approximation `root` as the average of `X` and `N / X`.

- Check if the absolute difference between `root` and `X` is less than `L`. If it is, return `root` as the approximate square root.

- Update `X` to the new `root` for further refinement.

Step 4: Prompt the user to input the number `N` for which the square root is needed and the tolerance level `L`.

Step 5: Call `newtons\_method\_sqrt` with `N` and `L`, storing the result in `sqrt\_approximation`.

Step 6: Print the approximate square root of `N`.

(a)

n=int(input("Enter n:"))

lst=[]

for i in range(2,n):

    count=0

    for x in range(2,int(i/2+1)):

        if i%x==0:

            count+=1

            break

    if count==0:

        lst.append(i)

print(lst)

(b)

n=int(input("Print the value of n:"))

m=int(input("Enter the value of m:"))

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

   print(n,"\*", i, "=",n\*i)

( c )

n=int(input("Enter the number of rows:"))

space=n

for i in range(n+1):

    for k in range(space):

        print(" ",end="")

    num=1

    for j in range(i):

       print(i,end=" ")

       num=num+1

    print()

    space-=1

space1=1

for i in range(n-1,0,-1):

    for k in range(space1):

        print(" ",end="")

    num=1

    for j in range(i):

        print(i,end=" ")

        num=num+1

    print()

    space1+=1

(D)

a=int(input("Enter the first integer:"))

b=int(input("Enter the second integer:"))

while b!=0:

  a,b=b,a%b

print("The GCD of two numbers is",a)

(e)

import math

sum=0

lst=[]

n=int(input("Enter the number of terms:"))

t1=0

t2=1

lst.append(t2)

for i in range(n):

  c=t1+t2

  t1=t2

  t2=c

  lst.append(c)

num=0

for i in lst:

  sum+=math.pow(-1,i+1)\*math.pow(i,num+1)

  num+=1

print("The sum of the series is:",sum)

(f)

num=int(input("Enter the number"))

sum=0

newnum=0

while(num!=0):

   newnum=int(num%10)

   sum+=newnum

   num=int(num/10)

print("The sum of the digits is:",sum)

(g)

def newtons\_method\_sqrt(N, L):

    X = N

    while True:

        root = 0.5 \* (X + N / X)

        if abs(root - X) < L:

            return root

        X = root

N = int(input("Enter the value of N: "))

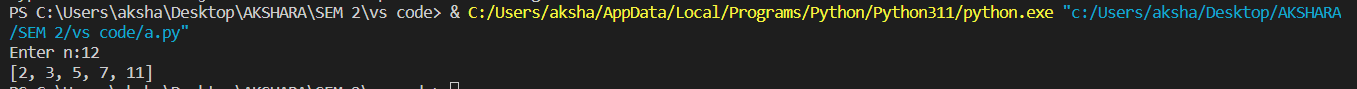
L = float(input("Enter the tolerance level L (<1): "))

sqrt\_approximation = newtons\_method\_sqrt(N, L)

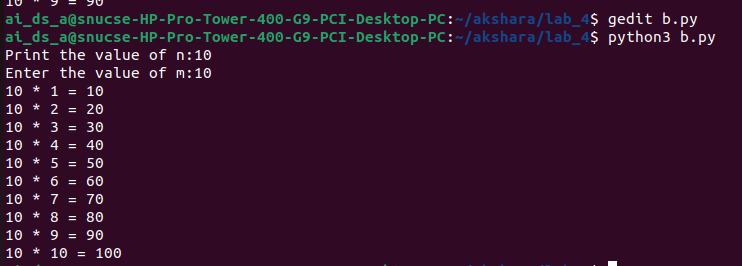
print(f"The square root is {N} is {sqrt\_approximation}")

Screenshot of Output:

(a)



(b)

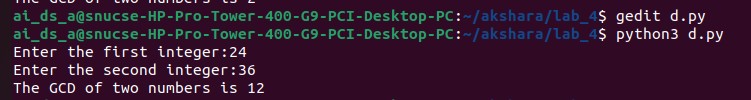


( c )

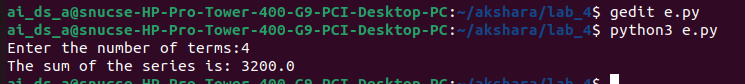
A black screen with yellow and green text

Description automatically generated

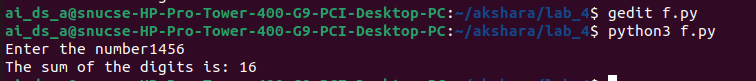
(d)



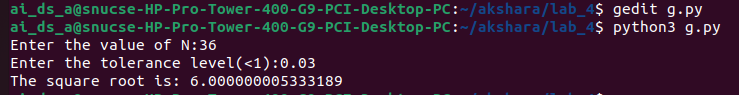
(e)



(f)



(g)



Result:

Thus, programs have been written and executed to explore the use of loops in Python.