|  |  |
| --- | --- |
| **Ex. 7** | **EXPLORING RECURSION** |
| **Date: 04/03/25** |  |

Aim:

To explore recursions in Python by writing recursive functions for the following:

1. Find the factorial of a given integer.
2. Find the value of a number, x, raised to the power of another number, y.
3. Write a menu driven program to generate the following sequences up to N terms by writing a recursive function for each:
   1. Fibonacci series
   2. Arithmetic progression, given the starting number and the common difference
   3. Geometric progression, given the starting number and the common ratio
   4. Harmonic progression, given the starting number and the common difference
   5. Triangular number series
4. Display the binary equivalent of a decimal number.
5. Find the largest digit in a given integer.
6. Compute 𝑛𝐶𝑟.
7. Display the sequence of steps to be followed to solve the Tower of Hanoi problem, given the number of disks, N.

Algorithm:

(a)

STEP 1: Prompt the user to input a number.

STEP 2: Define the function `fact` to calculate the factorial:

- If the number is 0, return 1, as the factorial of 0 is 1.

- Otherwise, return the product of the number and the factorial of the number minus one (recursive call).

STEP 3: Invoke the `fact` function with the user's input and print the result, which is the factorial of the given number.

(b)

STEP 1: Prompt the user to input a base number (`x`) and an exponent (`y`).

STEP 2: Define the function `power` to compute the power recursively:

- If the exponent `y` is 0, return 1, since any number to the power of 0 is 1.

- Otherwise, return the base number `x` multiplied by the result of the `power` function called with `x` and `y-1`.

STEP 3: Call the `power` function with the user's inputs for `x` and `y`, and print the result.

(c)

STEP 1: Display a menu with options for generating Fibonacci series, arithmetic progression (AP), geometric progression (GP), harmonic progression (HP), triangular numbers, and an exit option.

STEP 2: Prompt the user to enter their choice.

STEP 3: Based on the user's choice:

-If choice is 1:rompt for the number of terms and display the Fibonacci series up to that number.

- If choice is 2: Prompt for the first term, common difference, and number of terms, then display the arithmetic progression.

- If choice is 3:Prompt for the first term, common ratio, and number of terms, then display the geometric progression.

- If choice is 4: Prompt for the first term, common difference, and number of terms, then display the harmonic progression.

- If choice is 5: Prompt for `n` and display triangular numbers up to `n`.

- If choice is 6: Break the loop and exit the program.

- For invalid choices: Print an error message and show the menu again for a valid selection.

STEP 4: Continuously loop back to STEP 1 unless the user selects the option to exit (choice 6). This allows multiple operations without restarting the program

(D)

STEP 1: Prompt the user to enter a decimal number.

STEP 2: Define the function `binary` to perform the conversion:

- Check if the number is 0. If so, return immediately since the base case is the last call for non-zero values.

- Compute the remainder of the number divided by 2 (this gives the current binary digit).

- Print the remainder (binary digit) without moving to a new line.

- Recursively call the `binary` function with the quotient of the number divided by 2 (i.e., num // 2) to process the next higher binary digit.

STEP 3: Invoke the `binary` function with the user's input number

(e)

STEP 1: Prompt the user to enter a decimal number.

STEP 2: Define the function `largest` to determine the largest digit:

- Check if the current number num is zero. If so, return the largest found digit (`lar`).

- Extract the last digit of the number using the modulus operator (`num % 10`).

- If the extracted digit is larger than the current largest (`lar`), update `lar`.

- Recursively call the `largest` function with the quotient of the number (`num // 10`) to process the next digit and the updated largest digit.

STEP 3: Invoke the `largest` function with the user's input number and an initial `lar` value of 0.

.(f)

STEP 1: Prompt the user to input the total number of items, n , and the number of items to choose, r.

STEP 2: Define the `fact` function to calculate the factorial of a given number recursively

STEP 3: Define the `comb` function to calculate the combination of n items taken r at a time

STEP 4: Call the `comb` function with the values of n and r provided by the user, and print the result.

(g)

Step 1: Ask the user for the number of disks,N.

Step 2: Define the function `tower\_of\_hanoi` with parameters for the number of disks n,

Step 3: Execute `tower\_of\_hanoi` with N, 'A' as source, 'C' as destination, and 'B' as auxiliary.

Program:

(a)

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

def fact(num):

  if num==0:

     return 1

  else:

    return num\*fact(num-1)

print(fact(num))

(b)

x=int(input("Enter the number:"))

y=int(input("Enter the power:"))

def power(x,y):

    if(y==0):

      return 1

    else:

     return x\*pow(x,y-1)

print(power(x,y))

(c)

def fibonacci\_recursive(n, a=0, b=1):

    if n == 0:

       return

    else:

       print(a, end=' ')

       fibonacci\_recursive(n-1, b, a+b)

def print\_fibonacci\_series(n):

    print("Fibonacci Series up to", n, "terms:")

    fibonacci\_recursive(n)

def ap(an,d,n):

  if n==0:

   return 0

  else:

   print(an+d,end=" ")

   return ap(an+d,d,n-1)

def gp(a,r,n):

    if n==0:

       return 0

    else:

      print(a\*r,end=" ")

      return gp(a,r+r,n-1)

def hp(an,d,n):

   if n==0:

      return 0

   else:

      print(1/(an+d),end=" ")

      return hp(an+d,d,n-1)

def triangular\_number(n):

    if n == 1:

       return 1

    else:

      return n + triangular\_number(n - 1)

while(1):

  print("(1) Fibonacci Series")

  print("(2) Arithmetic Progression")

  print("(3) Geometric Progression")

  print("(4) Harmonic Progression")

  print("(5) Triangular Number series")

  print("(6) Exit")

choice=int(input("Enter your choice:"))

if choice==1:

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

   print\_fibonacci\_series(n)

elif choice==2:

  an=int(input("Enter the first term:"))

  d=int(input("Enter the common difference:"))

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

  print(an,end=" ")

  ap(an,d,n)

elif choice==3:

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

   r=int(input("Enter the common ratio:"))

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

   print(a,end=" ")

   gp(a,r,n)

elif choice==4:

  an=int(input("Enter the first term:"))

  d=int(input("Enter the common difference:"))

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

  hp(an,d,n)

elif choice==5:

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

  for i in range(1, n + 1):

     print(triangular\_number(i), end=' ')

elif choice==6:

   break

else:

  print("Enter a valid option")

(d)

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

def binary(num):

    sum=0

    if num==0:

       return 1

    else:

      sum=num%2

      print(sum,end="")

      return binary(num//2)

(e)

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

lar=0

def largest(num,lar):

    if(num==0):

      return lar

    else:

      number=num%10

      if(number>lar):

         lar=number

         return largest(num//10,lar)

print(largest(num,lar))

(f)

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

r=int(input("Enter the value of R:"))

sum=0

def fact(num):

    if num==0:

       return 1

    else:

       return num\*fact(num-1)

def comb(n,r):

   if n==0:

     return 1

   else:

     return fact(n)/(fact(r)\*fact(n-r))

print(comb(n,r))

(g)

def tower\_of\_hanoi(n, s, d,a):

   if n == 1:

      print(f"Move disk 1 from {s} to {d}")

      return

tower\_of\_hanoi(n-1, s, a, d)

print(f"Move disk {n} from {s} to {d}")

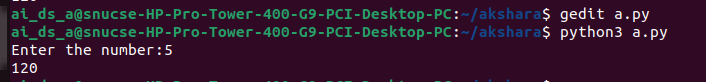
tower\_of\_hanoi(n-1, a, d, s)

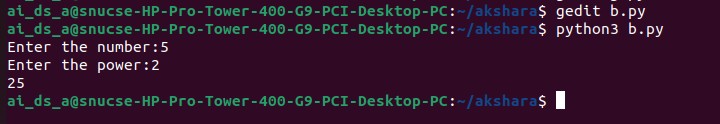
N = int(input("Enter the number of disks:"))

tower\_of\_hanoi(N, 'A', 'C', 'B')

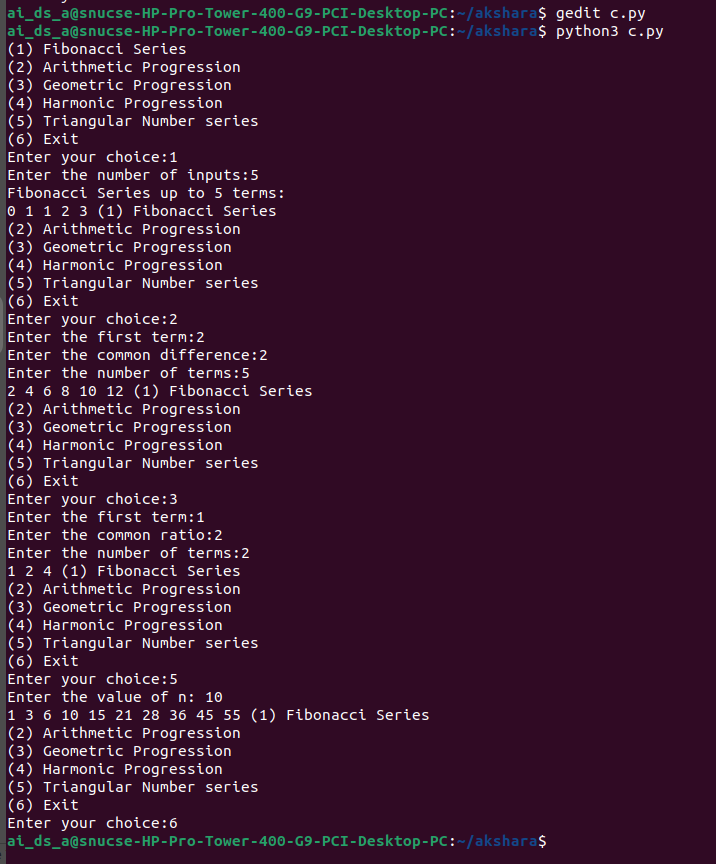
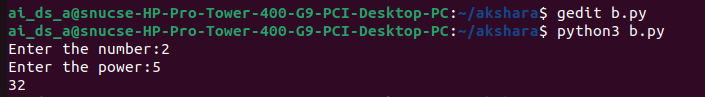
Screenshot of Output:

(a)

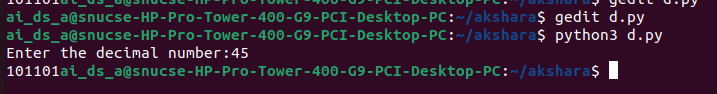


****(b)

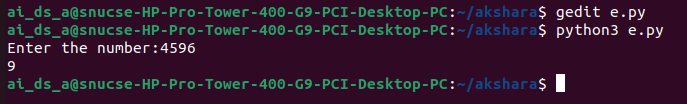
(c)



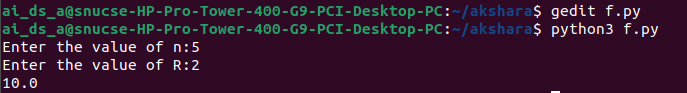
(d)

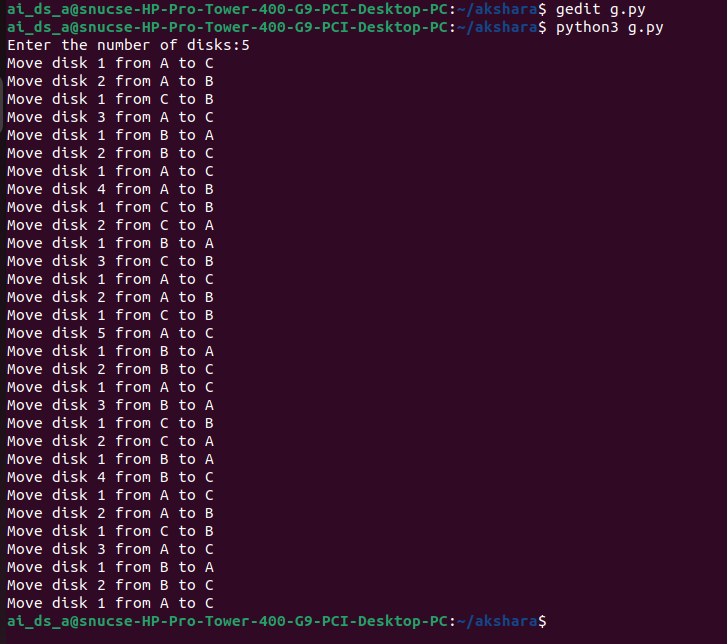


(e)



(f)



****(g)

Result:

Thus, programs have been written and executed to explore recursive functions in Python.