Experiment 04

Write a python program to understand User defined and Anonymous functions.

Roll No.	01
Name	Aamir Ansari
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO2: Interpret advanced data types and functions in python

Aim: Write a Python program to understand User defined and Anonymous functions.

Introduction:

Functions

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result
Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

Creating a Function
In Python a function is defined using the **def** keyword

```
def my_function():
    print("Hello from the other side!")
```

Calling a Function
To call a function, use the function name followed by parenthesis:

```
def my_function():
  print("Hello from the other side!")
# calling the function
my_function()
```

```
#output
Hello from the other side!
```

Arguments
Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
The following example has a function with one argument (name). When the function is called, we pass along a first name, which is used inside the function to print the full name

```
def my_function(name):
  print(name + " Ansari")

my_function("Aamir")
my_function("Ram")
my_function("Rahim")
```

Return statement: The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a none object is returned.
The return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

```
return [expression_list]
```

Examples of function:

```
def square_value(num):
    return num**2

print(square_value(5))
# prints 25
print(square_value(-1))
#prints 1
```

In Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

```
def func(x):
    x[0] = 20
    lst = [10, 11, 12, 13, 14, 15]
    func(lst)
    print(lst)
 #Output: [20, 11, 12, 13, 14, 15]
```

When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken. For example, consider the below program.

```
def myFun(x):
    x = [20, 30, 40]
    lst = [10, 11, 12, 13, 14, 15]
    myFun(lst)
    print(lst)
 #Output: [10, 11, 12, 13, 14, 15]
```

Another example to demonstrate that the reference link is broken if we assign a new value (inside the function).

```
def myFun(x): x = 20
    x = 10
    myFun(x)
    print(x)
#Output: 10
```

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

```
def printme( str ):
        "This prints a passed string into this function"
        print str
        return;
printme()
```

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

```
def printme( str ):
        "This prints a passed string into this function"
        print str
        return;
printme( str = "My string")
#Output: My String
```

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo( name, age = 35 ):
        "This prints a passed info into this function"
        print "Name: ", name Name:
        print "Age ", age
        return;
printinfo( age=50, name="miki" )
printinfo( name="miki" )
#Output: Name: Miki Age 50
#Output: Name: Miki Age 35
```

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax:

```
def functionname([formal_args,] *var_args_tuple ):
        "function_docstring"
        function_suite
        return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all non keyword

variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
def printinfo( arg1, *vartuple ):
        "This prints a variable passed arguments"
        print "Output is: "
        print arg1
        for var in vartuple:
                print var
        return;
printinfo( 10 )
printinfo( 70, 60, 50 )
# Output is : 10
# Output is : 50
# 60
# 70
```

Possible to define one function inside another function:

```
def outer(num1):
        def inner_increment(num1):
                return num1 + 1
        num2 = inner_increment(num1)
print(num1, num2)
outer(10)
#Output: 10 11
```

Possible to pass function as parameter to another function:

```
def display(fun):
        return ‘Hi’+fun
        def message():
                return ‘How are you?’
print(display(message())
#Output: Hi How are you?
```

Function can return other functions:

```
def display ():
        def message():
                return ‘How are you?’
        return message
fun=display()
print(fun())
```

#Output: How are you?

Scope and Lifetime of variables: Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope. The lifetime of a variable is the period throughout which the variable exists in the memory. The lifetime of variables inside a function is as long as the function executes.
They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.Here is an example to illustrate the scope of a variable inside a function.

```
def my_func(): #Output: Value inside function: 10
    x = 10
    print("Value inside function:",x)
    x = 20
my_func()
print("Value outside function:",x)
#Output: Value inside function: 10
#       Value outside function: 20
```

Here, we can see that the value of x is 20 initially. Even though the function my_func() changed the value of x to 10, it did not affect the value outside the function. This is because the variable x inside the function is different (local to the function) from the one outside. Although they have the same names, they are two different variables with different scopes. On the other hand, variables outside of the function are visible from inside. They have a global scope.

Anonymous Function: These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.

1. These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.
2. An anonymous function cannot be a direct call to print because lambda requires an expression
3. Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
4. Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax:

lambda [arg1 [,arg2,.....argn]]:expression

Following is the examples to show how lambda form of function works:

```
sum = lambda arg1, arg2: arg1 + arg2;
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
#Output: Value of total: 30
#        Value of total: 40
```

We can even use functions along with lambda functions.

```
def cube(x):
    return x*x*x 343
cube_v2 = lambda x : x*x*x
print(cube(7))
print(cube_v2(7))
#Output: 343
```

As we have mentioned earlier, the advantage of the lambda operator can be seen when it is used in combination with the map() function. map() is a function which takes two arguments: r = map(func, seq).

```
my_list = [1, 5, 4, 6, 8]
new_list = list(map(lambda x: x * 2, my_list)
print(new_list)
#Output: [1, 5, 4, 6, 8]
```

The function filter(function, sequence) offers an elegant way to filter out all the elements of a sequence, for which the function returns True. i.e. an item will be produced by the iterator result of filter(function, sequence) if item is included in the sequence "sequence" and if function(item) returns True.

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(filter(lambda x: (x%2 == 0), my_list)
print(new_list)
#Output: [4, 6, 8, 12]
```

The reduce() function in Python takes in a function and a list as an argument. The function is called with a lambda function and an iterable and a new reduced result is returned. This performs a repetitive operation over the pairs of the iterable. The reduce() function belongs to the functools module.

```
from functools import reduce
list1 = [5, 8, 10, 20, 50, 100]
```

```
    sum = reduce((lambda x, y: x + y), list1)
    print (sum)
#Output: 193
```

Recursion concept: The term Recursion can be defined as the process of defining something in terms of itself. In simple words, it is a process in which a function calls itself directly or indirectly.

Syntax:

```
    def func(): ←
                | (recursive call)
    func() ←-----
```

Example of recursive function is:

```
    def recursive_fibonacci(n):
        if n <= 1:
            return n
        else:
            return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2)) 3
    n_terms = 5
    if n_terms <= 0:
        print("Invalid input! Please input a positive value")
    else
        print("Fibonacci series:")
    for i in range(n_terms):
        print(recursive_fibonacci(i))
#Output: 0
# 1
# 1
# 2
# 3
```

Tail Recursion: A unique type of recursion where the last procedure of a function is a recursive call. The recursion may be automated away by performing the request in the current stack frame and returning the output instead of generating a new stack frame. The tail-recursion may be optimized by the compiler which makes it better than non-tail recursive functions.

```
    def Recur_facto(n):
    if (n == 0):
    return 1
    return n * Recur_facto(n-1)
    print(Recur_facto(6))
#Output: 720
```

Results

1. Python Program To Display Powers of 2 Using Anonymous Function.

Code:

```
numerOfTerms = 10
result = list(map(lambda x: 2 ** x, range(numerOfTerms)))
print("Number of terms are:",numerOfTerms)
for i in range(numerOfTerms):
    print("2 raised to power",i,"is",result[i])
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 1_Power_Anonymous.py
Number of terms are: 10
2 raised to power 0 is 1
2 raised to power 1 is 2
2 raised to power 2 is 4
2 raised to power 3 is 8
2 raised to power 4 is 16
2 raised to power 5 is 32
2 raised to power 6 is 64
2 raised to power 7 is 128
2 raised to power 8 is 256
2 raised to power 9 is 512
```

2. Python Program to Find Numbers Divisible by Another Number using Anonymous Function

Code:

```
my_list = [12, 65, 54, 39, 102, 339, 221,]
result = list(filter(lambda x: (x % 13 == 0), my_list))
print("Numbers divisible by 13 are",result)
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 2_Fibonacci_Anonymous.py
Numbers divisible by 13 are [65, 39, 221]
```

3. Python Program to Convert Decimal to Binary, Octal and Hexadecimal using Anonymous Function

Code:

```
num1 = int(input("Enter a decimal number: "))
bin=lambda x : format(x,'b')
print("Equivalent binary number is",bin(num1))
```

```
num2 = int(input("Enter a decimal number: "))
bin=lambda x : format(x,'o')
print("Equivalent octal number is",oct(num2))

num3 = int(input("Enter a decimal number: "))
bin=lambda x : format(x,'x')
print("Equivalent hexadecimal number is",hex(num3))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 3_Decimal-conversion_Anonymous.py
Enter a decimal number: 15
Equivalent binary number is 1111
Enter a decimal number: 2
Equivalent octal number is 0o2
Enter a decimal number: 17
Equivalent hexadecimal number is 0x11

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 3_Decimal-conversion_Anonymous.py
Enter a decimal number: 8
Equivalent binary number is 1000
Enter a decimal number: 8
Equivalent octal number is 0o10
Enter a decimal number: 8
Equivalent hexadecimal number is 0x8
```

4. Python Program to Find ASCII Value of Character

Code:

```
char = input("Enter a character: ")
print("The ASCII value of '" + char + "' is", ord(char))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 4_ASCII.py
Enter a character: 3
The ASCII value of '3' is 51

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 4_ASCII.py
Enter a character: /
The ASCII value of '/' is 47

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 4_ASCII.py
Enter a character: @
The ASCII value of '@' is 64

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 4_ASCII.py
Enter a character: L
The ASCII value of 'L' is 76
```

5. Python Program to Find HCF or GCD

Code:

```
def computeGCD(x, y):
   while(y):
       x, y = y, x % y
   return x

a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))

print ("The gcd of",a,"and",b,"is : ",end="")
print (computeGCD(a,b))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 5_HCF.py
Enter the first number: 18
Enter the second number: 9
The gcd of 18 and 9 is : 9

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 5_HCF.py
Enter the first number: 22
Enter the second number: 1
The gcd of 22 and 1 is : 1

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 5_HCF.py
Enter the first number: 35
Enter the second number: 5
The gcd of 35 and 5 is : 5

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 5_HCF.py
Enter the first number: 81
Enter the second number: 72
The gcd of 81 and 72 is : 9
```

6. Python Program to Find LCM

Code:

```
def getGcd(x, y):
   while(y):
       x, y = y, x % y
   return x
```

```
def getLcm(x, y):
    lcm = (x*y) // getGcd(x,y)
    return lcm

num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

print("The L.C.M. is", getLcm(num1, num2))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 6_LCM.py
Enter the first number: 6
Enter the second number: 3
The L.C.M. is 6

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 6_LCM.py
Enter the first number: 81
Enter the second number: 72
The L.C.M. is 648

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 6_LCM.py
Enter the first number: 56
Enter the second number: 72
The L.C.M. is 504
```

7. Python Program to Find Factors of Number Using Anonymous Function

Code:

```
num = int(input("Enter a number: "))
my_list = []

for i in range(1, num + 1):
    my_list.append(i)

new_list = list(filter(lambda x: (num % x == 0), my_list))

print("The factors of",num,"are:",new_list)
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 7_Factors_Anonymous.py
Enter a number: 32
The factors of 32 are: [1, 2, 4, 8, 16, 32]

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 7_Factors_Anonymous.py
Enter a number: 27
The factors of 27 are: [1, 3, 9, 27]

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 7_Factors_Anonymous.py
Enter a number: 81
The factors of 81 are: [1, 3, 9, 27, 81]

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 7_Factors_Anonymous.py
Enter a number: 79
The factors of 79 are: [1, 79]
```

8. Python Program to Make a Simple Calculator

Code:

```
# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    # Take input from the user
```

```
choice = input("Enter choice(1/2/3/4): ")

# Check if choice is one of the four options
if choice in ('1', '2', '3', '4'):
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    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))
    break
else:
    print("Invalid Input")
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 8_Calculator.py
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

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 8_Calculator.py
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 2
Enter first number: 7
Enter second number: 10
7.0 - 10.0 = -3.0
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 8_Calculator.py
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 3
Enter first number: 12
Enter second number: 5
12.0 * 5.0 = 60.0

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 8_Calculator.py
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 4
Enter first number: 96
Enter second number: 45
96.0 / 45.0 = 2.1333333333333333
```

9. Python Program to Display Calendar

Code:

```
import calendar

yy = int(input("Enter year: "))
mm = int(input("Enter month: "))

print("\n")
print(calendar.month(yy, mm))
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 9_Calendar.py
Enter year: 2020
Enter month: 4


     April 2020
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

10. Python Program to Display Fibonacci Sequence Using Recursion

Code:

```
def recur_fibo(n):
    if n <= 1:
        return n
    else:
        return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = 10

if nterms <= 0:
    print("Please enter a positive integer")
else:
    print("Fibonacci sequence:")

for i in range(nterms):
    print(recur_fibo(i))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 10_Fibonacci_Recursion.py
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
```

11. Python Program to Find Sum of Natural Numbers Using Recursion

Code:

```
def recur_sum(n):
    if n <= 1:
        return n
    else:
        return n + recur_sum(n-1)

num = int(input("Enter a number: "))
```

```
if (num < 0):
   print ("Enter a positive number")
else:
   print ("The sum is",recur_sum(num))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 11_Sum-natural_Recursion.py
Enter a number: 12
The sum is 78

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 11_Sum-natural_Recursion.py
Enter a number: 55
The sum is 1540

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 11_Sum-natural_Recursion.py
Enter a number: -5
Enter a positive number
```

12. Python Program to Find Factorial of Number Using Recursion

Code:

```
def recur_factorial(n):
   if n == 1:
      return n
   else:
      return n*recur_factorial(n-1)

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

if num < 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of", num, "is", recur_factorial(num))
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 12_Factorial_Recursion.py
Enter a number: 5
The factorial of 5 is 120

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 12_Factorial_Recursion.py
Enter a number: 6
The factorial of 6 is 720

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 12_Factorial_Recursion.py
Enter a number: -5
Sorry, factorial does not exist for negative numbers

E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 12_Factorial_Recursion.py
Enter a number: 0
The factorial of 0 is 1
```

13. Python Program to Convert Decimal to Binary Using Recursion

Code:

```
def convertToBinary(n):
    if n > 1:
        convertToBinary(n//2)
    print(n % 2,end = '')

dec = int(input("Enter a decimal number: "))

convertToBinary(dec)
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 13_Decimal-Binary_Recursion.py
Enter a decimal number: 8
1000
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 13_Decimal-Binary_Recursion.py
Enter a decimal number: 5
101
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 13_Decimal-Binary_Recursion.py
Enter a decimal number: 15
1111
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 13_Decimal-Binary_Recursion.py
Enter a decimal number: 55
110111
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>
```

14. Write a Menu driven program in python to implement a simple banking application.
Application should read the customer name, account number, initial balance, rate of interest, contact number and address field etc. Application should have following methods.
1. createAccount()
2. deposit()
3. withdraw()
4. computeInterest()
5. displayBalance()

Code:

```
class Bank_Account:
    def __init__(self):
        self.balance=0
        print("Hello!!! Welcome to the Deposit & Withdrawal Machine")

    def deposit(self):
        amount=float(input("Enter amount to be Deposited: "))
        self.balance += amount
        print("\n Amount Deposited:",amount)

    def withdraw(self):
        amount = float(input("Enter amount to be Withdrawn: "))
        if self.balance>=amount:
            self.balance-=amount
            print("\n You Withdrew:", amount)
        else:
            print("\n Insufficient balance  ")

    def display(self):
        print("\n Net Available Balance=",self.balance)

print("""
Welcome to Bank Management System:
1. Create Account
2. Deposit
3. Withdraw
4. Rate of Interest
5. Display Amount
6. Exit
""")
```

```
while True:
    print("\n")
    choice = input("Enter choice(1/2/3/4/5/6) :  ")
    print("\n")
    if choice in ('1', '2', '3', '4', '5', '6'):
        if choice == '1':
            name = input("Enter your name: ")
            account_no = int(input("Enter your account number: "))
            contact_no = int(input("Enter your phone number: "))
            address_field = input("Enter your address: ")
            s = Bank_Account()
        elif choice == '2':
            s.deposit()
        elif choice == '3':
            s.withdraw()
        elif choice == '4':
            s.computeinterest()
        elif choice == '5':
            s.display()
        else:
            exit()
    else:
        print("Invalid Option")
```

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 14_Bank.py

Welcome to Bank Management System:
1. Create Account
2. Deposit
3. Withdraw
4. Rate of Interest
5. Display Amount
6. Exit



Enter choice(1/2/3/4/5/6) :  1


Enter your name: Aamir
Enter your account number: 42069
Enter your phone number: 885585858
Enter your address: 221B, Baker street
Hello!!! Welcome to the Deposit & Withdrawal Machine
```

```
Enter choice(1/2/3/4/5/6) :  2


Enter amount to be Deposited: 2000

 Amount Deposited: 2000.0


Enter choice(1/2/3/4/5/6) :  3


Enter amount to be Withdrawn: 500

 You Withdrew: 500.0


Enter choice(1/2/3/4/5/6) :  4


Enter rate of interest: 5
New Balance: 9000.0


Enter choice(1/2/3/4/5/6) :  5



 Net Available Balance= 9000.0


Enter choice(1/2/3/4/5/6) :  6
```

15. Write a menu driven code in python which will read a number and should implement the following methods

Code:

```
def factorial(n):
    if n == 1:
        return n
    else:
        return n*factorial(n-1)

def reverse(n, r):
    if n==0:
        return r
    else:
        return reverse(n//10, r*10 + n%10)

def testArmstrong(n):
    order = len(str(n))
    sum = 0
    temp = n
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10
    return(sum)

def testPalindrome(n):
    rev = 0
    while n > 0:
        dig = n % 10
        rev = rev * 10 + dig
        n = n // 10
    return(rev)

def testPrime(n):
    flag = 0
    if n > 1:
        for i in range(2, n):
            if (n % i) == 0:
                flag = 1
            break
```

```
    return(flag)

def fibonacciSeries(n):
    if n <= 1:
        return n
    else:
        return(fibonacciSeries(n-1) + fibonacciSeries(n-2))

################################################################################
####

while True:
    print("""
    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series
    """)
    choice = input("Enter choice(1/2/3/4/5/6): ")

    if choice in ('1', '2', '3', '4', '5', '6'):
        if choice == '1':
            num = int(input("Enter a number: "))
            print("The factorial of", num, "is", factorial(num))

        elif choice == '2':
            num = int(input("Enter a number: "))
            reversed_number = reverse(num,0)
            print("Reverse of %d is %d" %(num, reversed_number))

        elif choice == '3':
            num = int(input("Enter a number: "))
            new_num = testArmstrong(num)
            if num == new_num:
                print(num,"is an Armstrong number")
            else:
                print(num,"is not an Armstrong number")
```

```
    elif choice == '4':
        num = int(input("Enter a number: "))
        new_num = testPalindrome(num)
        if num == new_num:
            print(num,"is a Palindrome number")
        else:
            print(num,"is not a Palindrome number")

    elif choice == '5':
        num = int(input("Enter a number: "))
        flag = testPrime(num)
        if flag == 0:
            print(num,"is a Prime number")
        else:
            print(num,"is not a Prime number")
    elif choice == '6':
        num = int(input("Enter a number: "))
        if num <= 0:
            print("Please enter a positive integer")
        else:
            print("Fibonacci sequence:")
        for i in range(num):
            print(fibonacciSeries(i))
    else:
        exit()
else:
    print("Invalid Option")
```

Output:

```
E:\Sem-4\Lab_Assignments\Python_Lab\Experiment_04\Code>python 15_MenuDriven.py

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 1
Enter a number: 6
The factorial of 6 is 720

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 2
Enter a number: 12345
Reverse of 12345 is 54321

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 3
Enter a number: 156
156 is not an Armstrong number
```

```
Enter choice(1/2/3/4/5/6): 3
Enter a number: 153
153 is an Armstrong number

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 4
Enter a number: 123
123 is not a Palindrome number

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 4
Enter a number: 12321
12321 is a Palindrome number

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 5
Enter a number: 81
81 is a Prime number
```

```
    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 5
Enter a number: 88
88 is not a Prime number

    Select operation:
    1. Factorial"
    2. Reverse"
    3. Test Armstrong number"
    4. Test Palindrome number"
    5. Test Prime number"
    6. Fibonacci Series

Enter choice(1/2/3/4/5/6): 6
Enter a number: 5
Fibonacci sequence:
0
1
1
2
3
```

Conclusion:

Hence, we have successfully understood and used the anonymous functions, user defined functions and recursion by solving various problems