1. What is a lambda function in Python, and how does it differ from a regular function?

A lambda function is a small anonymous function. A lambda function can take any number of arguments.

Lambda functions are anonymous so they cannot be referred anywhere else in the code and it cannot contain any statements and it returns a function object which can be assigned to any variable.

Regular functions on the other hand break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organised and manageable. They can be called and used anywhere we want.

1. Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?

Yes, lambda function can have multiple arguments.

(lambda x, y: x + y)(2, 3) this function can be used to find the sum of two numbers.

1. How are lambda functions typically used in Python? Provide an example use case.

The power of lambda is better shown when you use them as an anonymous function inside another function.

def myfunc(n):  
  return lambda a : a \* n  
  
mytripler = myfunc(3)  
  
print(mytripler(11))

This function triples the number passed as an argument.

1. What are the advantages and limitations of lambda functions compared to regular functions in Python?

Advantages:

They are anonymous so they can be created anywhere in the code and it cannot contain any statements and it returns a function object which can be assigned to any variable.

It can take any number of arguments.

Limitations:

As it is anonymous it can be referred any where else in the code.

1. Are lambda functions in Python able to access variables defined outside of their own scope?

Explain with an example.

Yes. they can access variables defined outside of their own scope.

Ex:

x **=** 5

def create\_lambda():

   return lambda: x

 my\_lambda **=** create\_lambda()

print(my\_lambda())

1. Write a lambda function to calculate the square of a given number.

Square=lambda x:x\*x

Square(2)

1. Create a lambda function to find the maximum value in a list of integers.

m=lambda x:max(x)

l=[1,5,9,10]

max(l)

1. Implement a lambda function to filter out all the even numbers from a list of integers.

fibo = [0,1,1,2,3,5,8,13,21,34,55]

print("List of fibonacci values :",fibo)

# filtering even values using lambda function

evenFibo = list(filter(lambda n:n%2==0,fibo))

print("List of even fibonacci values :",evenFibo)

1. Write a lambda function to sort a list of strings in ascending order based on the length of each string.

students = ['Prem', 'Shivang','Shahnail', 'Arun',‘raj']

students.sort(key=lambda name: len(name))

print(students)

1. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.

commlist=lambda x,y: set(x)&set(y)

list1=[3,5,8,10]

list2=[9,5,2,10]

commlist(list1,list2)

1. Write a recursive function to calculate the factorial of a given positive integer.

def factorial(n):

if n==1 or n==0:

return 1

else:

return (n\*factorial(n-1))

1. Implement a recursive function to compute the nth Fibonacci number.

def Fibonacci(n):

if n<= 0:

print("Incorrect input")

# First Fibonacci number is 0

elif n == 1:

return 0

# Second Fibonacci number is 1

elif n == 2:

return 1

else:

return Fibonacci(n-1)+Fibonacci(n-2)

1. Create a recursive function to find the sum of all the elements in a given list.

def listsum(l):

ln=len(l)

if ln>1:

return l[0]+listsum(l[1:])

else:

return l[0]

1. Write a recursive function to determine whether a given string is a palindrome.

def chckpalindrome(txt):

def reverse(txt):

ln=len(txt)

if ln>1:

return reverse(txt[1:])+txt[0]

else:

return txt

print(reverse(txt))

if txt==reverse(txt):

print("Is a palindrome")

else:

print("Is not a palindrome")

1. Implement a recursive function to find the greatest common divisor (GCD) of two positive integers.

def gcd(x , y):

if y == 0:

return x

else:

return gcd(y, x % y)