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

Lambda function is an anonymous one line function without any name. It is defined using the ‘lambda’ keyword. Lambda functions do not have any return statement, the result of the expression is automatically returned by the lambda function. Whereas regular function should be defined with name and should be callable.

Lambda functions are limited to a single expression, whereas regular functions have multiple statements and have more complex code.

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

YES lambda functions have multiple arguments, we can define multiple arguments by separate commas.

Ex

sum=lambda a,b,c:a+b+c

print(sum(1,2,3))

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

Lambda functions are commonly used with higher order functions like map,filter,reduce

Here is the example use case - filtering a list based on the condition

li=[1,2,3,4,5,6,7,8,8,9,88,0]

try :

even=list(filter(lambda x:x%2==0,li))

print(even)

except Exception as e:

print("Exception occured",e)

else:

print("no exception")

Output - [2, 4, 6, 8, 8, 88, 0]

no exception

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

**Advantages**

1.Lambda functions allow you to define small one line function without any name

2.Lambda functions are used in combination of higher order function like map,reduce,filter

3.Lambda functions can be easily combined or nested within other functions to create complex logic

**Limitations**

1.Lambda functions are limited to single expression, which means they cannot contain multiple statements

2.Lambda functions do not support doc strings, since no proper documentation.

3.limited debugging since they don’t have name its harder to identify and debug issues in lambda function

5. Are lambda functions in Python able to access variables defined outside of their own scope? Explain with an example.

Yes Lambda functions are able to access the variables which are defined outside of their own scope.

def lambdaownscope() :

x=10

sum=lambda y:x+y

return sum

a=lambdaownscope()

print(a(5))

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

givennumber=int(input())

square=lambda x:x\*\*2

print(square(givennumber))

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

li=[1,2,5,8,-1,2,9]

maximumelement=lambda listvalues:max(listvalues)

print(maximumelement(li))

Output: 9

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

li=[1,2,3,4,5,6,7,8,8,9,88,0]

try :

even=list(filter(lambda x:x%2==0,li))

print(even)

except Exception as e:

print("Exception occured",e)

output : [2, 4, 6, 8, 8, 88, 0]

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

string.

List=["manoj","jeyasri","nova","Bharat"]

Sorted=sorted(List,key=lambda x:len(x),reverse=False)

print(Sorted)

Output : ['nova', 'manoj', 'Bharat', 'jeyasri']

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

list1 = [1, 2, 3, 4, 5]

list2 = [4, 5, 6, 7, 8]

Common\_elements=list(filter(lambda x:x in list1,list2))

print(Common\_elements)

Output : [4,5]

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

def factorialofnumber(n) :

if n==0 or n==1:

return 1

else:

return n \* factorialofnumber(n-1)

print(factorialofnumber(int(input("Enter the number"))))

Input =5

Output=120

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

def fibonacci(n):

if n <= 0:

return "Invalid input. n must be a positive integer."

elif n == 1 or n == 2:

return 1

else:

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

result = fibonacci(6)

print(result) # Output: 8

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

def recursive\_sum(lst):

if len(lst) == 0:

return 0

else:

return lst[0] + recursive\_sum(lst[1:])

numbers = [1, 2, 3, 4, 5]

result = recursive\_sum(numbers)

print(result) # Output: 15

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

def is\_palindrome(string):

if len(string) <= 1:

return True

elif string[0] == string[-1]:

return is\_palindrome(string[1:-1])

else:

return False

print(is\_palindrome("racecar")) # Output: True

15. Implement a recursive function to find the greatest common divisor (GCD) of two positiveintegers.

def gcd(a, b):

if b == 0:

return a

else:

return gcd(b, a % b)

# Test cases

print(gcd(24, 36)) # Output: 12