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

Ans- Lambda functions are typically used when you need a simple, one-time function and don't want to define a regular function using the def keyword. They are commonly used in situations where a function is required as an argument to another function, such as in functional programming or when using higher-order functions like map(), filter(), or reduce().

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

Them?

Ans- Yes, a lambda function in Python can have multiple arguments. You can define and use them by specifying the arguments separated by commas after the lambda keyword.

multiply = lambda x, y: x \* y

result = multiply(5, 3)

print(result) # Output: 15

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

Ans- Lambda functions in Python are commonly used in situations where a function is required as an argument to another function, such as in functional programming or when using higher-order functions like map(), filter(), or reduce().

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

sqr\_num=list(map(lambda x:x\*\*2,num))

sqr\_num

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

Python?

Ans- Lambda functions in Python have several advantages and limitations compared to regular functions:

Advantages of Lambda Functions:

Concise Syntax: Lambda functions allow you to define small, one-time functions in a compact and inline manner, without the need for a formal function definition.

Readability: In some cases, lambda functions can enhance the readability of code, especially when used with higher-order functions like map(), filter(), or reduce(), where a simple operation needs to be applied to each element.

Function as an Argument: Lambda functions are often used as arguments to other functions, providing flexibility and convenience in functional programming scenarios.

Limitations of Lambda Functions:

Single Expression: Lambda functions are limited to a single expression, which means they cannot contain multiple statements, control structures (like if-else or loops), or complex logic. Regular functions have no such limitation.

Lack of Name: Lambda functions are anonymous and don't have a formal name. This can make it harder to debug or trace the origin of a specific lambda function.

Limited Functionality: Due to their restricted syntax, lambda functions are not suitable for complex or large-scale operations. Regular functions provide more flexibility and can encapsulate more intricate logic.

Reduced Reusability: Lambda functions are typically designed for one-time or specific-use cases and are not intended for reuse. Regular functions, on the other hand, can be defined once and called multiple times from different parts of the code.

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

Explain with an example.

Ans- Yes, lambda functions in Python are able to access variables defined outside of their own scope. This concept is known as "lexical scoping" or "closure." When a lambda function is defined, it retains access to the variables in its enclosing scope, even after that scope has finished executing.

def multiplier(n):

return lambda x: x \* n

multiply\_by\_2 = multiplier(2)

multiply\_by\_3 = multiplier(3)

print(multiply\_by\_2(5)) # Output: 10

print(multiply\_by\_3(5)) # Output: 15

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

Ans- square=lambda x:x\*\*2

square(9)

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

Ans- List=[1,2,3,6,81]

maximum= lambda x:max(x)

maximum(List)

# print(maximum) # Output: 81

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

Ans- List=[1,2,3,6,81]

sorted=list(filter( lambda x:x%2==0,List))

sorted

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

String.

Ans- string= ['apple', 'banana', 'cherry', 'date', 'elderberry']

sorted\_x=sorted(string,key= lambda x:len(x))

sorted\_x

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

common elements between the two lists.

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

list2=[1,2,6,7,8,9,4,5]

intersection= lambda l1,l2: list(filter(lambda x: x in l1,l2))

result= intersection(list1,list2)

print(result)

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

Ans- def fac(x):

if(x>0):

if x in [0,1]:

return 1

else:

return fac(x-1)\*x

else:

return 0

fac(3)

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

Ans- def fab(x):

if(x>0):

if x in [0,1]:

return x

else:

return fab(x-1)+fab(x-2)

else:

return 0

fab(5)

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

Ans- def sum(x):

if len(x)==0:

return 0

else:

return x[0]+ sum(x[1:])

sum(list1)

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

Ans- def is\_palindrome(string):

if len(string) <= 1:

return True

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

return False

else:

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

# Example usage:

word = "level"

result = is\_palindrome(word)

print(result)

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

Ans- def gcd(a,b):

if b==0:

return a

else:

return gcd(b,a%b)

gcd(12,18)