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 that can have any number of arguments but can only have one expression. It is defined using the lambda keyword, and its syntax is limited to a single line.

Lambda functions are anonymous, meaning they don't have a name.

Lambda functions are typically used for simple, short tasks, while regular functions can be more complex and have multiple statements.

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

them?

Yes, You can define and use them by specifying the arguments separated by commas in the lambda expression.

add = lambda x, y: x + y

result = add(3, 5)

print(result)

Output: 8

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

Lambda functions are often used when you need a short function for a specific task, especially as an argument to higher-order functions like map(), filter(), and sort().

my\_list = [1, 2, 3, 4, 5]

squared\_list = list(map(lambda x: x\*\*2, my\_list))

print(squared\_list)

Output: [1, 4, 9, 16, 25]

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

Python?

Advantages:

Concise and useful for simple tasks.

Can be used as throwaway functions when needed quickly.

Ideal for functional programming paradigms.

Limitations:

Limited to single-line expressions.

Lack of name and docstrings makes them less readable for complex operations.

Not suitable for tasks requiring multiple statements or complex logic.

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

Explain with an example.

Yes, if the variable is in the global scope or within an enclosing function.

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

square = lambda x: x\*\*2

result = square(5)

print(result)

Output: 25

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

max\_val = lambda nums: max(nums)

my\_list = [3, 8, 1, 6, 9]

result = max\_val(my\_list)

print(result)

Output: 9

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

even\_filter = lambda nums: list(filter(lambda x: x % 2 == 0, nums))

my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

result = even\_filter(my\_list)

print(result)

Output: [2, 4, 6, 8, 10]

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

string.

sort\_by\_length = lambda strings: sorted(strings, key=lambda x: len(x))

my\_list = ["apple", "banana", "cherry", "dates"]

result = sort\_by\_length(my\_list)

print(result)

Output: ['apple', 'dates', 'banana', 'cherry']

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

common elements between the two lists.

common\_elements = lambda list1, list2: list(set(list1) & set(list2))

list1 = [1, 2, 3, 4]

list2 = [3, 4, 5, 6]

result = common\_elements(list1, list2)

print(result)

Output: [3, 4]

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

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

result = factorial(5)

print(result)

Output: 120

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

def fibonacci(n):

if n <= 0:

return "Invalid input"

elif n == 1:

return 0

elif n == 2:

return 1

else:

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

result = fibonacci(6)

print(result) # Output: 5

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

def sum\_list(nums):

if not nums:

return 0

return nums[0] + sum\_list(nums[1:])

my\_list = [1, 2, 3, 4, 5]

result = sum\_list(my\_list)

print(result) # Output: 15

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

def is\_palindrome(s):

if len(s) <= 1:

return True

if s[0] != s[-1]:

return False

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

text = "radar"

result = is\_palindrome(text)

print(result) # Output: True

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

def gcd(a, b):

if b == 0:

return a

return gcd(b, a % b)

result = gcd(60, 48)

print(result) # Output: 12