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

Lambda functions are different from regular functions in a few ways:

* Lambda functions are shorter. They can only contain a single expression, so they are much shorter than regular functions.
* Lambda functions are unnamed. They do not have a name, so they cannot be called directly.
* Lambda functions are more concise. They can be used inline, which makes them more concise than regular functions.

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

A lambda function in Python can have multiple arguments. You can define and use multiple arguments in a lambda function by separating them with commas in the function definition. Here's an example:

python code:

# Lambda function with multiple arguments

addition = lambda x, y: x + y

result = addition(5, 7)

print(result) # Output: 12

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

Lambda functions in Python are typically used in scenarios where a small, one-time function is needed, especially in combination with higher-order functions like map(), filter(), and reduce(). They provide a concise way to express simple logic without the need to define a named function separately.

Here's an example use case:

# Example

names = ["John Doe", "Alice Smith", "Bob Johnson", "Emily Davis"]

sorted\_names = sorted(names, key=lambda x: x.split()[-1])

print(sorted\_names)

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

Advantages of Lambda Functions:

Conciseness: Lambda functions allow you to express simple and short logic in a compact, one-line format.

Anonymous: Lambda functions are anonymous, meaning they don't require a name.

Function as a First-Class Object: Lambda functions can be treated as first-class objects in Python, just like regular functions.

Readability: Lambda functions can make code more readable in certain scenarios, especially when used with higher-order functions like map(), filter(), and reduce().

Limitations of Lambda Functions:

Single Expression: Lambda functions are limited to a single expression.

No Documentation Strings: Lambda functions do not support documentation strings (docstrings).

Limited Functionality: Lambda functions are designed for simplicity and brevity.

Readability in Complex Cases

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

Yes, lambda functions in Python can access variables defined outside of their own scope. They have access to variables from the enclosing scope, including global variables and variables defined in the surrounding 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.

numbers = [10, 5, 8, 12, 3]

max\_value = lambda nums: max(nums)

result = max\_value(numbers)

print(result) # Output: 12

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

numbers = [10, 5, 8, 12, 3, 7, 6, 9, 2]

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers) # Output: [10, 8, 12, 6, 2]

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

strings = ["apple", "banana", "orange", "kiwi", "grape"]

sorted\_strings = sorted(strings, key=lambda x: len(x))

print(sorted\_strings) # Output: ['kiwi', 'grape', 'apple', 'banana', 'orange']

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 factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

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

def fibonacci(n):

if n <= 0:

raise ValueError("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 list\_sum(lst):

if len(lst) == 0:

return 0

else:

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

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

result = list\_sum(my\_list)

print(result) # Output: 15

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

def is\_palindrome(string):

string = string.lower()

if len(string) <= 1:

return True

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

return False

else:

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

result = is\_palindrome("radar")

print(result) # Output: True

result = is\_palindrome("hello")

print(result) # Output: False

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

else:

return gcd(b, a % b)

result = gcd(48, 18)

print(result) # Output: 6

result = gcd(35, 14)

print(result) # Output: 7