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 in Python that can have any number of parameters but can only have one expression. Lambda functions are created using the `lambda` keyword, and they are typically used for short and simple operations.

- Lambda functions differ from regular functions in that they do not have a name and are usually used for one-time, throwaway tasks where creating a full-fledged function using `def` is not necessary.

Regular Function Example:

def add(a, b):

return a + b

Lambda Function Equivalent:

add = lambda a, b: a + b

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

Yes, a lambda function in Python can have multiple arguments. You can define and use them just like in a regular function, but the syntax is more concise.

Example of a lambda function with multiple arguments:

add = lambda a, b: a + b

print(add(2, 3)) # Output: 5

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

Lambda functions are typically used in situations where a small, simple function is needed temporarily or as an argument to other functions like `map()`, `filter()`, `sorted()`, etc.

Example use case of lambda function with `sorted()`:

names = ['Alice', 'Bob', 'Charlie', 'David', 'Eva']

sorted\_names = sorted(names, key=lambda x: len(x))

print(sorted\_names)

# Output: ['Bob', 'Eva', 'Alice', 'David', 'Charlie']

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

Advantages of lambda functions:

- Concise and can be written in a single line.

- Useful for one-time, short tasks.

- No need to assign a name, reducing clutter in the code.

Limitations of lambda functions:

- Can only consist of a single expression.

- Cannot contain statements or multiple lines of code.

- Lack of a name makes them less readable in complex scenarios.

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

Yes, lambda functions can access variables defined outside their own scope, just like regular functions. This is known as a closure.

Example:

def multiplier(factor):

return lambda x: x \* factor

multiply\_by\_3 = multiplier(3)

print(multiply\_by\_3(5))

# Output: 15

In this example, the lambda function `multiply\_by\_3` accesses the `factor` variable defined in the `multiplier` function's scope.

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

square = lambda x: x \*\* 2

print(square(5))

# Output: 25

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

numbers = [10, 5, 15, 25, 7]

max\_num = lambda nums: max(nums)

print(max\_num(numbers))

# Output: 25

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

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

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

print(even\_numbers(numbers))

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

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

names = ['Alice', 'Bob', 'Charlie', 'David', 'Eva']

sorted\_names = sorted(names, key=lambda x: len(x))

print(sorted\_names)

# Output: ['Bob', 'Eva', 'Alice', 'David', 'Charlie']

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 = lambda lst1, lst2: list(set(lst1) & set(lst2))

print(common\_elements(list1, list2)) # 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)

print(factorial(5))

# 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)

print(fibonacci(7))

# Output: 8

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

def recursive\_sum(nums):

if not nums:

return 0

else:

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

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

print(recursive\_sum(numbers))

# 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

else:

return s[0] == s[-1] and is\_palindrome(s[1:-1])

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

print(is\_palindrome("hello")) # 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)

print(gcd(24, 18))

# Output: 6