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

Ans. A lambda function in Python is a small anonymous function that can be defined in a single line using the lambda keyword. It is primarily used for creating simple throwaway functions without explicitly naming them.

The main difference between a lambda function and a regular function lies in their syntax and usage. Lambda functions are typically used for short quick calculations rather than for complex reusable code. They can only contain a single expression and automatically return the result of that expression. In contrast regular functions are defined using the def keyword have a name and can contain multiple statements and a return statement.

1. 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. To define and use multiple arguments in a lambda function you can use the syntax `lambda arg1 arg2 argn: expression`.

Ex: -

multiply = lambda x y: x \* y

print(multiply(5 3))

Output: 15

We can have any number of arguments in a lambda function by separating them with commas. For example:

Ex: -

addition = lambda a b c: a + b + c

print(addition(2 4 6))

Output: 12

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

Ans. Lambda functions in Python are anonymous functions that can be defined inline and used for short simple operations. They are typically used in situations where a small one-time function is needed without the need to assign a name to it. One common use case for lambda functions is in functional programming and with higher-order functions like `map() `filter() and `reduce()`. For example we can use a lambda function with `map()` to apply a specific operation on each element of a list:

numbers = [1 2 3 4 5]

squared\_numbers = list(map(lambda x: x\*\*2 numbers))

In this example the lambda function `lambda x: x\*\*2` is applied to each element `x` in the `numbers` list resulting in a new list `squared\_numbers` containing the squared values.

Lambda functions are also useful in sorting and defining custom criteria. They allow us to define a specific key function inline without the need to define a separate function.

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

Ans. Lambda functions also known as anonymous functions provide several advantages over regular functions in Python. Here are some key advantages:

* Lambda functions allow you to write simple one-line functions without the need for a complete function definition. This makes code more compact and readable.
* Lambda functions follow a minimalist syntax reducing the need for explicit return statements or function names. They are ideal for cases where a small single-use function is required.
* Lambda functions can be used wherever a function object is expected such as in higher-order functions like map filter and reduce(). They allow you to define and use small functions on-the-fly without the need for defining a named function separately.

some limitations to consider when using lambda functions:

* Lambda functions are limited to a single expression. They cannot contain multiple statements or complex logic making them unsuitable for longer or more intricate functions.
* Lambda functions don't support docstrings making it difficult to provide detailed documentation for the function's purpose and usage.
* While lambda functions can make code more concise they may sacrifice readability for complex logic. It can be challenging to understand the purpose and behavior of a lambda function at first glance without proper variable naming or comments.

1. 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 is because lambda functions have access to the variables in the enclosing scope which is known as the closure.

Input: -

def outer\_function():

x = 10

inner\_function = lambda y: x + y

return inner\_function

closure\_function = outer\_function() # Returns the lambda function

result = closure\_function(5) # Accesses x from the outer function's scope

print(result)

Output: -

15

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

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

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

Ans.

Input: -

nums = [3 9 2 5 1]

maximum = max\_value(nums)

print(maximum)

Output: -

9

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

Ans.

Input: -

numbers = [1 2 3 4 5 6 7 8 9 10]

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

print(filtered\_numbers)

Output: -

[2 4 6 8 10]

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

Ans.

Input: -

strings = ['apple 'banana 'pear 'orange']

sorted\_strings = sort\_strings(strings)

print(sorted\_strings)

output: -

['pear 'apple 'orange 'banana']

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

Ans.

Input: -

list1 = [1 2 3 4]

list2 = [3 4 5 6]

result = common\_elements(list1 list2)

print(result)

Output: -

[3 4]

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

Ans.

Input: -

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

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

Ans.

def fibonacci(n):

if n <= 0:

raise ValueError("Fibonacci sequence is only defined for positive integers.")

elif n == 1 or n == 2:

return 1

else:

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

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

Ans.

def find\_sum(lst):

if len(lst) == 0:

return 0

else:

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

my\_list = [1 2 3 4 5]

sum\_of\_list = find\_sum(my\_list)

print(sum\_of\_list)

Output: -

15

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

Ans.

def is\_palindrome(string):

# Base case: if the string is empty or contains only one character it is a palindrome

if len(string) <= 1:

return True

# Recursive case: compare the first and last characters and check if the remaining substring is a palindrome

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

# Remove the first and last characters and check the remaining substring

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

else:

return False

print(is\_palindrome("racecar") )

print(is\_palindrome("hello"))

print(is\_palindrome("madam"))

Output: -

True

False

True

1. 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)

num1 = 24

num2 = 36

print("GCD of num1 "and num2 "is gcd(num1 num2))

Output:

GCD of 24 and 36 is: 12