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 short and anonymous function defined using ‘lambda’ keyword. Its used for short and simple calculations and it does not require separate name like regular function.

e.g.

addition = lambda x, y: x + y

result = addition(5, 6)

print(result) # Output: 11

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 and you can use them like below:

e.g.

multiplication = lambda x, y: x \* y

result = multiplication (5, 6)

print(result) # Output: 30

division = lambda x, y: x / y

result = multiplication (10, 5)

print(result) # Output: 2

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

Ans: Lambda functions in python are typically used in situations where a short, one-time function is needed. Such as when passing a function as an argument to the function.

e.g.

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

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

print(even\_numbers)

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

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

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

Advantages:

* Concise syntax – lambda function allows you to write a function in single line which can make code more readable.
* Easy to use with higher order functions: lambda functions are often used with higher order functions such as map(), filter() and reduce().
* No need for a separate name

Limitations:

* Single expression: Lambda function allows you to contain only a single expression, which reduce their complexity
* Limited readability: Lambda functions can sometimes make code less readable, especially if they are too long or contain complex logic

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 can access variables defined outside of their own scope. Just like regular functions

e.g.

outside\_variable = 10

lambda\_function = lambda x: x + outside\_variable

result = lambda\_function(10)

print(result) # Output: 20

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

Ans:

Square = lambda x: x \*\* 2

result = Square(2)

print(result) # Output: 4

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

Ans: Here is lambda function that finds the maximum value in a list on integers:

max\_value = lambda lst: max(lst)

numbers = [5,4,7,8,10]

result= max\_value(numbers)

print(result) # Output: 10

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

Ans: Here is lambda function that filter out all the even numbers from a list:

Filter\_evn = lambda lst: list(filter( lambda x: x % 2 == 2, lst))

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

even\_numbers = Filter\_evn(numbers)

print(even\_number) # 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.

Ans: Here's a lambda function that sorts a list of strings in ascending order based on the length of each string:

Lambda\_function = lambda lst: sorted(lst, key = lambda x: len(x))

Strings = [“Apple”, “Banana”, “Cherry”, “Grapes”]

Sorted\_string = Lambda\_function(Strings)

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: here's a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists:

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

list1 = [1,4,5,7,6]

list2 = [2,4,1,6,8]

result = comman\_elements(list1, list2)

print(result)

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

Ans: here's a recursive function in Python to calculate the factorial of a given positive integer:

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

result = factorial(4)

print(result) # Output: 24

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

Ans: Here's a recursive function in Python to compute the nth Fibonacci number:

e.g.

def fibonacci(n):

if n <= 0:

return “Invalid input, please provide positive number.”

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.

Ans: here's a recursive function in Python 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: ])

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

result = list\_sum(numbers)

print(result) # Output: 15

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

Ans:

def is\_palindrome(s):

if len(s) <= 1:

return True

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

return False

else:

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

string1 = "radar"

string2 = "hello"

print(is\_palindrome(string1)) # Output: True

print(is\_palindrome(string2)) # Output: False

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)

result = gcd(10, 5)

print(result) # Output: 5