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

= The lambda functions can be used without any declaration in the namespace. The lambda functions defined as single-line functions.

These functions do not have parenthesis like the def defined functions but instead, take parameters after the lambda keyword.

There is no return keyword defined explicitly because the lambda function does return an object by default.

defined functions are commonly used because of their simplicity. The def defined functions do not return anything if not explicitly returned whereas the lambda function does return an object.

The def functions must be declared in the namespace.

The def functions can perform any python task including multiple conditions, nested conditions or loops of any level,raise an exceptions, etc.

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

= Lambda functions are defined using the keyword lambda. They can have any number of arguments but only one expression. A lambda function cannot contain any statements, and it returns a function object which can be assigned to any variable.

train\_frame['Sex'] = train\_frame['Sex'].

apply(lambda x : 1

if x =='male'

else 0)

Lambda functions are good for situations where you want to minimize lines of code as you can create function in one line of python code. It is not possible using def

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

= It allows you to supply a function as a parameter to a different function (for example, in map, filter, etc.).

Lambda functions can only contain a single expression and cannot contain multiple lines of code. They are designed for short and simple operations.

we defined a lambda function(upper) to convert a string to its upper case using [upper()](https://www.geeksforgeeks.org/python-string-upper/).

eg ; str1 = 'I Neuron'

1. upper = lambda string: string.upper()

print(upper(str1))

#Output = I NEURON

2. Max = lambda a, b : a if(a > b) else b

print(Max(1, 2))

# output = 2

3, Filter Function.

list = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final\_list = list(filter(lambda x: (x % 2 != 0), list))

print(final\_list)

# Output [5, 7, 97, 77, 23, 73, 61]

list = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

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

print(final\_list)

# Output = [10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

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

= Lambda functions can be a powerful tool for writing concise, readable, and efficient code.

As this is a single-line syntax, the function is much more readable. There's no need to search for the argument on another line.

lambda functions return the result of evaluating an expression. There is no need to use a return statement explicitly.

Regular functions return a value explicitly using a return statement. If no return statement is used, the function returns None.

Lambda functions in Python are generally faster than regular functions because they are implemented as anonymous functions and do not require a name lookup.

One of the main limitations is that lambda functions are limited to a single expression, meaning that they cannot contain multiple statements or complex.

being restricted to a single expression and having limited functionality compared to named functions.

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

= Lambda expressions can use variables defined in an outer scope. We refer to these lambdas as Capturing Lambdas. They can capture static variables, instance variables, and local variables, but only local variables must be final.

Lambda functions are used in map, filters and reduce functions.

A closure is a function object that remembers values in enclosing scopes even if they are not present in memory. When a lambda function accesses a variable from its surrounding scope, it "closes over" that variable,

def outer\_function(x):

y = 10

return lambda z : x+y+z

closure= outer\_function=(5)

result = closure (20)

print(result)

# Output = 35 (10+5+20)

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

= def square(n):

return n\*n

numbers = (1,2,3,4)

result=map(lamba x : x\*x, numbers)

print(tuples(result))

# Output = (1,4,9,16)

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

= from functools import reduce

list = [22, 36, 47, 2, 13]

reduce(lambda x, y: x if (x > y)

else y, list)

47

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

=  apply a lambda expression with the filter() method. The lambda expression (n -> n % 2 == 0) checks if a number is divisible by 2 (i.e., even).

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

# Output list initialisation

out = []

for num in list:

    # checking condition

    if num % 2 == 0:

        out.append(num)

# printing output

print(out)

# Output = [2,4]

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

= my\_numbers = [10, 8, 3, 22, 33, 7, 11, 100, 54]

#sort list in-place in ascending order

my\_numbers.sort()

#print modified list

print(my\_numbers)

#output

[3, 7, 8, 10, 11, 22, 33, 54, 100]

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 = [1,2,3,4,5,6]

list2 = [3, 5, 7, 9]

list(set(list1).intersection(list2))

[3, 5]

Or

def intersection(lst1, lst2):

    lst3 = [list(filter(lambda x: x in lst1, sublist)) for sublist in lst2]

    return lst3

lst1 = [1, 6, 7, 10, 13, 28, 32, 41, 58, 63]

lst2 = [[13, 17, 18, 21, 32], [7, 11, 13, 14, 28], [1, 5, 6, 8, 15, 16]]

print(intersection(lst1, lst2))

# Output [[13, 32], [7, 13, 28], [1, 6]]

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

= def factorial(n):

    # Checking the number

    # is 1 or 0 then

    # return 1

    # otherwise return

    # factorial

    if (n==1 or n==0):

        return 1

    else:

        return (n \* factorial(n - 1))

num = 5

print("number : ",num)

print("Factorial : ",factorial(num))

number : 5

Factorial : 120

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

= int fib(int n)

{

    int a = 0, b = 1, c, i;

    if (n == 0)

        return a;

    for (i = 2; i <= n; i++) {

        c = a + b;

        a = b;

        b = c;

    }

    return b;

}

int main()

{

    int n = 9;

    count << fib(n);

    return 0;

}

# Output 34.

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

= def recursive\_sum(my\_list, n):

if n == 0:

return my\_list[n]

else: return my\_list[n] + recursive\_sum(my\_list, n-1)

my\_list = [1, 2, 3, 4, 5] result = recursive\_sum(my\_list, len(my\_list)-1)

print(result)

# Output = 15.

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

= a=raw\_input("enter the string:")

b=len(a)

c=0

for i in range(b):

if a[i]==a[-(i+1)]:

c=c+1

if c==b:

print a,"is polindrome"

else:

print a,"is not polindrome"

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

= num1 = 36

num2 = 60

gcd = 1

for i in range(1, min(num1, num2)):

if num1 % i == 0 and num2 % i == 0:

gcd = i

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

# Output = GCD of 36 and 60 is 12