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

   It is a small, anonymous function defined without a name. Its syntax is:

   Lambda arguments: expression

   Eg: square = lambda x : x**2

   Regular function are suitable for more complex problems where flexibility is needed whereas lambda functions are concise and suitable for simple operations. They also known as single line functions.

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

   Yes, a lambda function can have multiple arguments. We can define and use multiple arguments by separating them with commas within the argument list.

   Eg: multiply = lambda x,y : x*y

   multiply(5,3)

   Output➔ 15

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

   Lambda functions are typically used in conjunction with higher order functions. These functions accept function as an argument. Few examples of such functions are **map(), filter()** and **reduce().** Lambda functions are convenient to use inline as an argument.

   Eg: a =[1,2,3,4,5]
   square= map(lambda x : x**2, a)
   print(list(square))

   output➔ [1,4,9,16,25]

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

   Advantages: Lambda functions are simple, inline functions. Simple problems can be solved within a line only. Lambda functions are more concise and have improved readability when logics are straight forward. Lambda functions facilitates function composition, means it allows combining with multiple lambda functions or with regular functions. Thus, able to solve more complex operations by chaining the functions.

   Disadvantages: lambda functions cannot be used for complex problems and where more flexibility is needed. Loops and condition expressions are less suitable as lambda expressions. Sometimes, they are very complex to understand when number of argument and complexity of function increases.

   Lambda functions are anonymous inline function, this lack of name and separate definition limits its reusability. Challenges in debugging and handling errors due to its complexity.

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 are able to access variable outside of their own scope. They have access to the variables from enclosing scope, including global variables and variables in the other functions.

Eg:

```
def func_1():
    x=10
    return lambda y : x+y

func_2 = func_1()
    print(func_2(10))
```

Output➔ 20

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 = [7, 12, 5, 9, 3, 5]
max_value = max(numbers, key=lambda x: x)
print(max_value)
```

output➔ 12

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

```
lst =[1,2,4,7,8]
even_number = list(filter(lambda x :  x%2 ==0 , lst))
print(even_number)
```

output ➔ [2,4,8]

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

```
lst = ["Bike", "cat","steamer", "aeroplane"]
sorted_str = sorted(lst, key = lambda x : len(x))
print(sorted_str)
```

output ➔ ['cat', 'Bike', 'steamer', 'aeroplane']

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

```
lst_1 = [1,3,4,5,8,9]
lst_2 = [1,2,3,2,4,5]

common_elements = filter(lambda x : x in lst_1, lst_2)
print(list(common_elements))
```

output➔ [1,3,4,5]

11. Write a recursive function 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)
```

```
# Example usage
number = 5
result = factorial(number)
print(result)
```

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

```
# nth fabonnaci number 1,1,2,3,5,8,13,21  ,...
def fabonnachi(n):
    if n<=0:
        return none
    elif n==1 or n==2:
        return 1
    else:
        return  fabonnachi(n-2) + fabonnachi(n-1)

fabonnachi(4)
output➔ 3
```

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

```
def sum_list(lst):
    if len(lst) == 0:
        return 0
    else:
        return lst[0] + sum_list(lst[1:])

# Example usage
my_list = [1, 2, 3, 4, 5]
result = sum_list(my_list)
print(result)
output➔ 15
```

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

```
def is_palindrome(string):
    if len(string)<=1:
        return True
    elif string[0]!=string[-1]:
        return False
    else:
        return is_palindrome(string[1:-1])

is_palindrome("level")   #output➔ True
is_palindrome("label")   # 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)

# Example usage
num1 = 36
```

```
num2 = 48
result = gcd(num1, num2)
print(result)  # Output➔ 12
```