1. A lambda function in Python is an anonymous function defined using the lambda keyword. It is a way to create small, one-line functions without explicitly defining a named function. Lambda functions differ from regular functions in that they can only contain a single expression and have no associated function name.

2. Yes, a lambda function in Python can have multiple arguments. Multiple arguments are defined by separating them with commas within the lambda function's parameter list. For example, a lambda function with multiple arguments can be defined as follows: `lambda x, y: x + y`.

3. Lambda functions are typically used in Python when a small, one-time function is required, especially as an argument to higher-order functions like `map()`, `filter()`, or `reduce()`. They provide a concise way to define simple functions on the fly without the need for separate function definitions. For example, using a lambda function with `map()` to square each element of a list: `squared\_values = map(lambda x: x\*\*2, my\_list)`.

4. The advantages of lambda functions compared to regular functions in Python include their conciseness and ability to be defined inline without the need for a separate function definition. Lambda functions are also useful in functional programming paradigms. However, lambda functions have limitations such as being restricted to a single expression and lacking the ability to contain complex logic or statements.

5. Yes, lambda functions in Python can access variables defined outside of their own scope. They have access to variables in the enclosing scope, including global variables. Here's an example: `x = 10; my\_lambda = lambda y: x + y`.

6. Lambda function to calculate the square of a given number: `square = lambda x: x\*\*2`.

7. Lambda function to find the maximum value in a list of integers: `max\_value = lambda lst: max(lst)`.

8. Lambda function to filter out all the even numbers from a list of integers: `even\_numbers = lambda lst: list(filter(lambda x: x % 2 == 0, lst))`.

9. Lambda function to sort a list of strings in ascending order based on the length of each string: `sorted\_strings = lambda lst: sorted(lst, key=lambda x: len(x))`.

10. Lambda function to find the common elements between two lists: `common\_elements = lambda lst1, lst2: list(filter(lambda x: x in lst2, lst1))`.

11. 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)

12. Recursive function to compute the nth Fibonacci number:

def fibonacci(n):

if n <= 1:

return n

else:

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

13. Recursive function to find the sum of all the elements in a given list:

def sum\_list(lst):

if not lst:

return 0

else:

return lst[0] + sum\_list(lst[1:])

14. Recursive function to determine whether a given string is a palindrome:

def is\_palindrome(s):

if len(s) <= 1:

return True

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

return False

else:

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

15. 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)