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

A lambda function in Python is a small and anonymous function defined using the lambda keyword. It can have any number of arguments but can only have one expression. Lambda functions are often used for short, simple operations where defining a full-fledged function using def would be unnecessary and less readable.

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

Yes, a lambda function in Python can have multiple arguments. Lambda functions can take any number of arguments, just like regular functions. The syntax for defining a lambda function with multiple arguments is as follows:

```
# Lambda function to calculate the sum of two numbers
add = lambda x, y: x + y

# Test the lambda function
result = add(5, 10)
print(result)  # Output: 15
```

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

Lambda functions are typically used in Python for simple, short-lived operations where defining a full-fledged named function using def would be unnecessary or less readable. They are particularly useful when you need to pass a small function as an argument to other functions, like in higher-order functions such as map(), filter(), and sorted(), or when you need to create quick, throwaway functions.
Example:

```
students = [("Alice", 25), ("Bob", 20), ("Charlie", 30), ("David", 22)]

# Sort the list of students based on their ages using a lambda function
sorted_students = sorted(students, key=lambda student: student[1])

print(sorted_students)
# Output: [('Bob', 20), ('David', 22), ('Alice', 25), ('Charlie', 30)]
```

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

lambda functions offer advantages in terms of conciseness, readability, and convenience for simple operations and quick throwaway functions. However, they have limitations in terms of functionality and debugging capabilities compared to regular functions. The choice between using lambda functions and regular functions depends on the complexity and purpose of the function you need to define.

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 can access variables defined outside of their own scope. This is because lambda functions inherit the scope of the code block in which they are defined, just like regular functions.

```
def outer_function():
    x = 10

    # Define a lambda function inside the outer_function
    inner_lambda = lambda y: x + y

    return inner_lambda

# Call the outer_function and get the inner lambda function
lambda_func = outer_function()

# Call the lambda function with an argument
result = lambda_func(5)
print(result)  # Output: 15
```

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

```
square = lambda x: x ** 2
```

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

```
max_value = lambda lst: max(lst)
```

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

```
filter_even = lambda lst: list(filter(lambda x: x % 2 == 0, lst))
```

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

```
sort_by_length = lambda lst: sorted(lst, key=lambda x: len(x))
```

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

```
find_common_elements = lambda lst1, lst2: list(filter(lambda x: x in lst2,
lst1))
```

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)
```

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

```
def fibonacci(n):
    if n <= 0:
        return None  # Invalid input
    elif n == 1:
        return 0  # The first Fibonacci number is 0
    elif n == 2:
        return 1  # The second Fibonacci number is 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

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

```
def recursive_sum(lst):
    if not lst:
        return 0  # Base case: an empty list has a sum of 0
    else:
        return lst[0] + recursive_sum(lst[1:])
```

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

```
def is_palindrome(s):
    # Base case: An empty string or a string with one character is a
palindrome
    if len(s) <= 1:
        return True
    # Recursive case: Check if the first and last characters are the same
    if s[0] == s[-1]:
        # Recur for the substring with the first and last characters
removed
        return is_palindrome(s[1:-1])
    else:
        return False
```

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

```
def gcd(a, b):
    # Base case: If b is 0, then GCD is a
    if b == 0:
        return a
    # Recursive case: Find GCD using the Euclidean algorithm
    return gcd(b, a % b)
```