**Question 1**:

A lambda function is a small anonymous function defined using the lambda keyword. Lambda functions are useful for creating short, one-line functions.

Syntax: ***lambda arguments: expression***

Example:

***square = lambda x: x \*\* 2***

***print(square(5)) # Output: 25***

Differences between normal function and lambda function:

* Lambda functions are defined using a single line, while regular functions are defined using the def keyword and can span multiple lines.
* Lambda functions are anonymous, meaning they don't have a name like regular functions do. They are often used where a small function is needed for a short duration and doesn't need to be referred to by name later.
* Lambda functions automatically return the result of the expression they evaluate, whereas regular functions require an explicit return statement to return a value.

**Question 2**:

Yes, lambda function can have multiple arguments which can be separated by commas.

Syntax: ***lambda arg1, arg2, arg3, ...: expression***

Example:

***add = lambda x, y, z: x + y + z***

***print(add(3, 5, 7))***

**Question 3**:

Lambda functions in Python are typically used in situations where a small, anonymous function is needed for a short duration or as an argument to higher-order functions. They are particularly useful in functional programming constructs like map(), filter(), and sorted(), where a function is applied to each element of an iterable or used to define sorting criteria.

students = [('Alice', 85), ('Bob', 90), ('Charlie', 80), ('David', 95)]  
sorted\_students = sorted(students, key=lambda x: x[1])  
  
print(sorted\_students)  
# Output: [('Charlie', 80), ('Alice', 85), ('Bob', 90), ('David', 95)]

**Question 4**:

Advantages of Lambda function:

* Lambda functions allow you to define small, anonymous functions in a single line of code. This can lead to more compact and readable code.
* This is useful when you need a small function for a short duration and don't want to clutter your code with unnecessary function names.
* Lambda functions are commonly used in functional programming constructs like map(), filter(), and sorted().

Limitations of lambda function:

* Lambda functions can only contain a single expression.
* Lambda functions can also make it less readable, especially for beginners or when used excessively.
* Lambda functions cannot contain statements like return, pass, or assert, and they cannot include multiple lines of code.

**Question 5**:

Yes, lambda functions in Python can access variables defined outside of their own scope. This is possible because lambda functions have access to the variables in the enclosing scope.

def outer\_function():  
 x = 10  
 return lambda y: x + y  
  
add\_to\_x = outer\_function()  
result = add\_to\_x(5)  
  
print(result) # Output: 15

**Question 6**:

square = lambda x: x \*\* 2  
print(square(5)) # Output: 25

**Question 7**:

arr = [5, 10, 3, 8, 15, 7]  
max\_value = lambda x: max(x)  
result = max\_value(arr)  
print("Maximum value in the list:", result) # Output: 15

**Question 8**:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
filter\_even = lambda x: x % 2 == 0  
even\_numbers = list(filter(filter\_even, numbers))  
  
print("Even numbers in the list:", even\_numbers) # Output: [2, 4, 6, 8, 10]

**Question 9**:

strings = ["apple", "banana", "kiwi", "orange", "strawberry"]  
sort\_by\_length = lambda x: len(x)  
sorted\_strings = sorted(strings, key=sort\_by\_length)  
  
print("Sorted list of strings by length:", sorted\_strings)  
# Output: ['kiwi', 'apple', 'banana', 'orange', 'strawberry']

**Question 10**:

list1 = [1, 2, 3, 4, 5]  
list2 = [4, 5, 6, 7, 8]  
  
common\_elements = lambda x, y: list(filter(lambda elem: elem in y, x))  
  
result = common\_elements(list1, list2)  
print("Common elements between the two lists:", result) # Output: [4, 5]

**Question 11**:

def factorial(n):  
 if n == 1 or n==0:  
 return 1  
 else:  
 return n \* factorial(n - 1)  
  
print(factorial(5)) # Output: 120

**Question 12**:

def fibonacci(n):  
 if n <= 1:  
 return n  
 else:  
 return fibonacci(n - 1) + fibonacci(n - 2)  
  
number = 6  
result = fibonacci(number)  
print("The", number, "th Fibonacci number is:", result)

**Question 13**:

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("Sum of all elements in the list:", result)

**Question 14**:

def is\_palindrome(s):  
 if len(s) <= 1:  
 return True  
 else:  
 return s[0] == s[-1] and is\_palindrome(s[1:-1])  
  
string1 = "radar"  
string2 = "hello"  
print(string1, "is a palindrome:", is\_palindrome(string1))  
print(string2, "is a palindrome:", is\_palindrome(string2))

**Question 15**:

def gcd(a, b):  
 if b == 0:  
 return a  
 else:  
 return gcd(b, a % b)  
  
number1 = 48  
number2 = 18  
result = gcd(number1, number2)  
print("The GCD of", number1, "and", number2, "is:", result)