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

Ans: In Python, a lambda function, also known as an anonymous function, is a way to define small, one-line functions without using the **def** keyword. Lambda functions are typically used for simple operations and can be created and used inline without assigning them to a variable.

syntax : lambda arguments: expression

Lambda functions and regular functions have some key differences:

1. **Syntax:** Lambda functions are defined using a more concise syntax in a single line, while regular functions are defined using the **def** keyword and can span multiple lines.
2. **Name:** Lambda functions are anonymous, meaning they don't have a specific name assigned to them. They are typically assigned to a variable or used directly in function calls. On the other hand, regular functions are named and can be called by their defined name.
3. **Function Body:** Lambda functions are limited to a single expression. They can't contain multiple statements or complex logic. Regular functions can have multiple statements, including control flow statements (e.g., **if**, **for**, **while**), local variable declarations, and more complex logic.
4. **Return Statement:** In lambda functions, the result is automatically returned based on the evaluated expression. In regular functions, the **return** statement is used to explicitly define what value should be returned from the function.
5. **Scope:** Lambda functions have access to variables in the enclosing scope, just like regular functions. This includes global variables, variables defined in the outer function if used inside one, and any variables passed as arguments.
6. Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?

Ans: Yes, a Lambda function in Python can have multiple arguments.

multiply = lambda x, y: x \* y

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

Ans: Lambda functions are typically used for functional programming, sorting, for small fucntions, filtering.

For ex: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even\_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even\_numbers))   
# Output: [2, 4, 6, 8, 10]

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

Ans: Advantages of Lambda Functions:

* **Conciseness**: Lambda functions allow you to define small, anonymous functions in a compact and concise manner, typically within a single line of code
* **Readability**
* **Inline Usage**: Lambda functions can be used directly at the point where they are needed, without the need to define them elsewhere in the code.

Limitations of Lambda Functions:

* **Single Expression**
* **Limited Functionality**
* **Limited Naming**

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

Ans: Yes, lambda functions in Python can access variables defined outside of their own scope. Lambda functions have access to variables from the surrounding scope, including global variables and variables defined in the enclosing function. This behavior is known as lexical scoping or closure.

def outer\_function():

outer\_variable = 10

# Lambda function accessing outer\_variable

lambda\_function = lambda x: x + outer\_variable

return lambda\_function

# Call the outer function and assign the lambda function to a variable

my\_lambda = outer\_function()

# Call the lambda function with an argument

result = my\_lambda(5)

print(result) # Output: 15

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

Ans: square = lambda x: x\*x

square(5) #output: 25

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

Ans: numbers =[4,56,67,8,99,98]

max\_list = lambda list: max(list)

max\_value = max\_list(numbers)

max\_value

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

Ans: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers) # Output: [2, 4, 6, 8, 10]

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

Ans: strings = ["apple", "banana", "cherry", "date", "elderberry"]

sorted\_strings = sorted(strings, key=lambda x: len(x))

print(sorted\_strings) # Output: ['date', 'apple', 'cherry', 'banana', 'elderberry']

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

Ans: common\_elements = lambda lst1, lst2: [x for x in lst1 if x in lst2]

lst1 = [1,2,3,4,6]

lst2 = [1,5,6,8,9]

common\_elements(lst1,lst2)

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

Ans: def factorial(x):

y = 1

for i in range(1,x+1):

y = y\*i

return y

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

Ans: def fibonacci(n):

if n <= 0:

raise ValueError("n must be a positive integer.")

elif n == 1:

return 0

elif n == 2

return 1

else:

fib\_n\_minus\_1 = 1

fib\_n\_minus\_2 = 0

fib\_n = 0

for i in range(3, n + 1):

fib\_n = fib\_n\_minus\_1 + fib\_n\_minus\_2

fib\_n\_minus\_2 = fib\_n\_minus\_1

fib\_n\_minus\_1 = fib\_n

return fib\_n

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

Ans:

def sum\_of\_list(n):

if type(n) != list:

raise ValueError("n must be a list.")

else:

sum=0

for i in range(0,len(n)):

if type(n[i])!=int:

raise ValueError("list should not contain anything other than integer values.")

else:

sum=sum+n[i]

return sum

sum\_of\_list([1,2,3])

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

Ans:

def is\_palindrome(string):

clean\_chars = []

for ch in string:

if ch.isalnum():

clean\_chars.append(ch.lower())

clean\_string = ''.join(clean\_chars)

reversed\_string = clean\_string[::-1]

return clean\_string == reversed\_string

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

Ans:

def gcd(a, b):

if b == 0:

return a

else:

return gcd(b, a % b)