Assignment

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

Ans : In Python, a lambda function is an anonymous function that can have any number of arguments but can only have one expression. Lambda functions are defined using the lambda keyword, followed by the arguments and the expression.

The main differences between lambda functions and regular functions are:

1. Syntax: Lambda functions use a compact syntax with the lambda keyword, while regular functions are defined using the def keyword.
2. Name: Lambda functions are anonymous; they don't have a name. Therefore, they are typically used when the function is simple and won't be referred to elsewhere in the code.
3. Number of expressions: Lambda functions can only have one expression. This means they cannot contain multiple statements or complex logic like regular functions.
4. Return value: Lambda functions automatically return the value of the expression without needing an explicit return statement. In contrast, regular functions require a return statement to return a value.
5. Scope: Lambda functions can access variables from the surrounding scope, similar to regular functions and other Python objects. However, it's essential to understand the concept of closures when dealing with lambda functions and accessing variables from the enclosing scope.

2. 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.

lambda arg1, arg2, ..., argN: expression

Eg - add = lambda x, y: x + y

addition = add(5, 3)

print(result)

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

Ans : Lambda functions in Python are typically used for small, simple tasks where defining a full-fledged named function is not required.

Lambda functions are used for sorting. Eg –

students = [{'name': 'Alice', 'age': 25},

{'name': 'Bob', 'age': 20},

{'name': 'Charlie', 'age': 22}]

sorted\_students = sorted(students, key=lambda x: x['age'])

print(sorted\_students)

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

Python?

Ans : Advantages of Lambda Functions:

1. Concise Syntax: Lambda functions have a compact syntax, making them suitable for short and simple operations.
2. Anonymous: Lambda functions are anonymous, meaning they don't require a name. This can be useful for one-time, throwaway functions where there is no need to define a function name explicitly.
3. First-class Citizens: Lambda functions are first-class citizens in Python, just like regular functions. They can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.
4. Functional Programming: Lambda functions are commonly used in functional programming paradigms, where functions are treated as data and can be manipulated like any other data type.

Limitations of Lambda Functions:

1. Single Expression: Lambda functions can only contain a single expression. This means they are not suitable for complex logic or multiple statements.
2. No Documentation: Lambda functions lack the ability to include documentation strings (docstrings) that provide details about the function's purpose, parameters, and behavior. Docstrings are a valuable way to document functions for clarity and maintainability.
3. Limited Readability: While lambda functions can be concise, they can also become less readable if they contain too much logic.
4. Debugging Challenges: When using lambda functions, debugging can be more challenging because they lack a name to identify them in tracebacks or error messages.
5. No Statements: Lambda functions can only contain expressions and not statements, so operations like assignments or print statements are not allowed.Top of Form

5. 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. They have access to variables from the surrounding environment, including global variables.

Eg - def outer\_function(x):

y = 10

lambda\_func = lambda z: x + y + z

return lambda\_func

my\_lambda = outer\_function(5)

result = my\_lambda(3)

print(result)

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

Ans : y = lambda x : x\*\*2

Print(y(5))

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

Ans : find\_max = lambda lst: max(lst)

numbers = [10, 5, 18, 3, 22, 7]

result = find\_max(numbers)

print(result)

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

Ans : filter\_even = lambda lst: list(filter(lambda x: x % 2 == 0, lst))

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

even\_numbers = filter\_even(numbers)

print(even\_numbers)

# Output: [2, 4, 6, 8, 10]

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

string.

Ans : sort\_by\_length = lambda lst: sorted(lst, key=lambda x: len(x))

words = ['apple', 'banana', 'orange', 'grape', 'kiwi']

sorted\_words = sort\_by\_length(words)

print(sorted\_words)

# Output: ['kiwi', 'grape', 'apple', 'banana', 'orange']

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

common elements between the two lists.

Ans : find\_common\_elements = lambda list1, list2: list(filter(lambda x: x in list2, list1))

list1 = [1, 2, 3, 4, 5]

list2 = [3, 4, 5, 6, 7]

common\_elements = find\_common\_elements(list1, list2)

print(common\_elements) # Output: [3, 4, 5]

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

Ans : def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

# Example usage

num = 5

result = factorial(num)

print(f"The factorial of {num} is: {result}")

# Output: The factorial of 5 is: 120

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

Ans : def fib(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

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

num = 5

result = fib(num)

print(f"The fibonacci of {num} is: {result}")

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

Ans : def rec\_sum(lst):

if not lst:

return 0

else:

return lst[0] + rec\_sum(lst[1:])

numbers = [1, 2, 3, 4, 5]

result = rec\_sum(numbers)

print(f"The sum of elements in the list is: {result}")

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

Ans : 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. 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)

num1 = 125

num2 = 50

result = gcd(num1, num2)

print(f"The GCD of {num1} and {num2} is: {result}")

# The GCD of 125 and 50 is: 25