Assignment 2

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

**Answer:** In Python, a lambda function is a small anonymous function that can be defined without a name. It is created using the **lambda** keyword and is typically used when you need a simple, one-line function without the need for a formal function definition.

Syntax: lambda arguments: expression.

Lambda functions use lambda keyword, followed by the arguments and a colon, and then the expression whereas Regular functions use **def** keyword, followed by the function name, arguments within parentheses, a colon, and the function body.

Anonymous: Lambda functions are anonymous functions, meaning they are not bound to a specific name. Regular functions have a defined name and can be called by that name whenever needed.

Limited Functionality: Lambda functions are limited to a single expression. Regular functions can include multiple statements and have more flexibility in terms of their functionality.

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

**Answer:** A lambda function in Python can have multiple arguments. You can define and use multiple arguments in a lambda function by separating them with commas in the argument list.

Ex:

product = lambda x, y: x \* y

print (product (3, 4))

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

**Answer:** Lambda functions in Python are typically used in situations where a small, anonymous function is required for a specific task. They are often used in combination with other functions or methods that accept function objects as arguments, such as map(), filter(), and sort().

**Ex:** numbers = [1, 2, 3, 4, 5]

squares = list (map (lambda x: x \*\* 2, numbers))

print(squares)

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

**Answer:**

**Advantages of Lambda Functions:**

* Simple functions with a compact syntax, eliminating the need for a formal function definition. This makes the code more concise and readable.
* Lambda functions are anonymous, meaning they don't require a specific name.
* Lambda functions are useful in functional programming paradigms where functions can be composed together. They can be easily combined with other functions like **map ()**, **filter ()**, and **reduce ()**.
* Lambda functions are evaluated and executed immediately when defined.

**Disadvantages of Lambda Functions:**

* Lambda functions are limited to a single expression only and not suitable for complex operations.
* Lambda functions are anonymous, so they can't be directly referenced or called by name.
* Lambda functions are designed for simple and concise operations and not for more complex functionality.

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

**Answer:**

Lambda functions in Python can access variables defined outside of their own scope. They have access to variables from the enclosing scope. This concept is known as "lexical scoping" or "closure."

**Ex:**

def outer\_function():

    x = 10

lambda\_func = lambda y: x + y

return lambda\_func

my\_lambda = outer\_function()

result = my\_lambda(5)

print(result)

**Output:** 15

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

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

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

**Answer:**

find\_max = lambda lst: max(lst)

numbers = [5, 2, 9, 1, 7]

result = find\_max(numbers)

print(result)

**Output:** 9

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

**Answer:**

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

number = [3, 2, 5, 9, 8 , 4, 12, 20, 10, 65, 66, 54, 45]

result = filter\_even(number)

print(result)

**Output :** [2, 8, 4, 12, 20, 10, 66, 54]

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

**Answer:**

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

player\_Names = ["Dhoni", "Virat", "Rahane", "Ruturaj", "Jadeja", "Conway", "Pathirana", "Shivam"]

result = sort\_by\_length(player\_Names)

print(result)

**Output:** ['Dhoni', 'Virat', 'Rahane', 'Jadeja', 'Conway', 'Shivam', 'Ruturaj', 'Pathirana']

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

**Answer:**

find\_common\_elements = lambda lst1, lst2: list(filter(lambda x: x in lst2, lst1))

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

list2 = [4, 5, 6, 7, 8]

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

print(result)

**Output:** [4, 5]

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

**Answer:**

def factorial(n):

if n == 1:

return 1

else:

return n \* factorial(n-1)

print(factorial(5))

**Output:** 120

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

**Answer:**

def fibonacci(n):

if n <= 1:

return n

else:

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

print(fibonacci(9))

**Output: 8**

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

**Answer:**

def recursive\_sum(lst):

if len(lst) == 0:

return 0

else:

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

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

result = recursive\_sum(numbers)

print(result)

**Output:** 15

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

**Answer:**

def is\_palindrome(s):

    if len(s) <= 1:

return True

else:

if s[0] == s[-1]:

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

else:

return False

string1 = "radar"

print(is\_palindrome(string1))

**Output:** True

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

**Answer:**

def gcd(a, b):

if b == 0:

return a

else:

return gcd(b, a % b)

a = 84

b = 18

result = gcd(a, b)

print(result)

**Output: 6**