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

Answer:

**Lambda functions,** also known as anonymous functions, are small, one-line functions that do

not have a name. lambda functions can take any number of arguments but can only have a

single expression.

**Syntax:**

lambda arguments: expression

**# Regular function to square a number**

**def square(x):**

**return x \*\* 2**

**# Lambda function to square a number**

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

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

**print(square\_lambda(5))**

Lambda functions are often used in situations where a small, simple function is required as an

argument to another function, such as in functional programming, when working with higher-order

functions like map(), filter(), or reduce().

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

Answer:

Yes, a lambda function in Python can have multiple arguments. The syntax for defining a lambda function with multiple arguments is as follows:

**lambda arg1, arg2, ..., argn: expression**

**multiply = lambda x, y: x \* y**

**result = multiply(3, 4)**

**print(result) # Output: 12**

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

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

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

**print(squared\_numbers) # Output: [1, 4, 9, 16, 25]**

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

Answer:

**Advantages of lambda functions compared to regular functions in Python:**

* Concise and readable for simple, one-line operations.
* Anonymous nature allows for temporary or inline function definitions.
* Enables function composition and works well with higher-order functions.

**Limitations of lambda functions compared to regular functions in Python:**

* Limited to a single expression, cannot contain multiple statements or complex control flow.
* Lack of documentation, making it harder for other developers to understand their purpose.
* Readability can be compromised for longer or complex expressions.

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

Answer:

Yes, lambda functions in Python are able to access variables defined outside of their own scope. This behavior is known as "lexical scoping" or "closure". When a lambda function is defined, it retains access to variables from the surrounding scope in which it was defined.

Program:

**def multiplier(n):**

**return lambda x: x \* n**

**multiply\_by\_2 = multiplier(2)**

**multiply\_by\_3 = multiplier(3)**

**print(multiply\_by\_2(5)) # Output: 10**

**print(multiply\_by\_3(5)) # Output: 15**

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

Answer:

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

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

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

Answer:

**maximum = lambda x: max(x)**

**print(maximum([1,2,3,4,5]))#Output: 5**

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

Answer:

**list\_item = [1,5,6,5,5,6,8,9,8,72,5,2,55,45]**

**filtered\_list= list(filter(lambda x:x%2==0,list\_item))**

**print(filtered\_list)#Ouput: [6, 6, 8, 8, 72, 2]**

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

Answer:

**names = ["Vikas","Anju","Sikarwar","Parmar","Sik"]**

**sorted\_names = sorted(names,key=lambda x:len(x))**

**print(sorted\_names)**

**Output: ['Sik', 'Anju', 'Vikas', 'Parmar', 'Sikarwar']**

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

Answer:

**lst1 = [15, 9, 10, 56, 23, 78, 5, 4, 9]**

**lst2 = [9, 4, 5, 36, 47, 26, 10, 45, 87]**

**common\_elements = lambda list1, list2: list(set(list1) & set(list2))**

**print(common\_elements(lst1,lst2))**

**#Ouput: [9, 10, 4, 5]**

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

Answer:

**def fact(n):**

**if n in [0,1]:**

**return 1**

**else:**

**result = n\*fact(n-1)**

**return result**

**Number = int(input(“Please provide the number to get the factorial of a number”))**

**print(fact(number))**

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

Answer:

**def fibonacci(n):**

**if n <=1:**

**return 0**

**elif n in [1,2]:**

**return 1**

**else:**

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

**print(fibonacci(5))#Output: 3**

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

**Answer:**

**def sum\_recursive(list\_item):**

**if len(list\_item) == 0:**

**return 0**

**else:**

**return list\_item[0]+sum\_recursive(list\_item[1:])**

**print(sum\_recursive([1,2,4]))**

**#Ouput: 7**

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

**Answer**:

**def is\_palindrome(string):**

**if len(string) <= 1:**

**return "String is palindrome"**

**# Check if the first and last characters are equal**

**if string[0] == string[-1]:**

**return is\_palindrome(string[1:-1])**

**# If the first and last characters are not equal, it's not a palindrome**

**return "this string is not palindrome"**

**text = input("enter the text to check the palindrome : ")**

**print(is\_palindrome(text))**

15. 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= int(input("Kindly enter the first value : "))**

**b= int(input("Kindly enter the Second value : "))**

**print(gcd(a,b))**