<u>Q 1.</u> What is a lambda function in Python, and how does it differ from a regular function?

<u>Answer</u> –

Lambda functions are unnamed functions defined using the *lambda* keyword. Lambda functions are different from regular functions in following ways

Naming - The regular functions have a name and lambda functions are anonymous.

Lines of code - The lambda functions are defined in a single expression, whereas the regular functions can have multiple lines of code.

Functionality - The lambda functions have limited functionality due to their concise structure. Whereas regular functions can have varied levels of functionality.

Example -

```
x = lambda a : a + 10

print(x(5))
```

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

<u>Answer</u> –

Yes, the lambda function can have multiple arguments. The arguments can be separated by a comma before the colon. Following we define a lambda function that calculates average of the arguments provided. The lambda function accepts n number of arguments. The user can *avg* to calculate the average for the varied number of values.

```
avg = lambda *args: sum(args) / len(args)
result = calculate_average(5, 10, 15)
print(result)  >> Output: 10.0
result = calculate_average(2, 4, 6, 8, 10)
print(result)  >> Output: 6.0
```

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

Answer –

The lambda functions are used when a small function is required for a particular operation. So instead of defining a complete function we use the single line lambda expression to get the job done.

Use case - Following we have a dictionary named *student_data.* We intend to sort it by the increasing order of the age of the student. The lambda function helps us sort the data by the age of the student.

```
student_data= [ {"name": "Akhil", "age": 20},
{"name": "Sam", "age": 18},
{"name": "Dravid", "age": 22},
{"name": "Virat", "age": 19} ]
student_data_sort = sorted(student_data, key=lambda x: x["age"])
print(student_data_sort )
```

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

Answer –

Advantages

- Concise - The syntax of the lambda function is concise as it can be defined in a single expression.
- Combination - The lambda functions can be combined with other lambda functions to create more complex operations.
- Inline use - The lambda function can be used directly as an argument to functions like *filter(), map()*.

Limitations

- Single expression. The lambda function is limited to a single line. They do not contain multiple lines of code, which limits its functionality.
- Naming - The lambda functions are anonymous. This makes it challenging during error handling and debugging
- Reusability - The lambda functions are designed for on the go use, meaning they are short lived. They are not intended to to used in multiple parts of the codet intended to be used at multiple parts of the code.

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

Answer –

Yes, Lambda functions can access the variables defined outside their own scope. Following we can see the variable x defined outside the scope of lambda function and the lambda function is able to access the variable x

```
a=1
b=2
x=3
sum = lambda a, b: a + b + x
print(sum(3,4)) >> Output =10
```

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

Answer –

```
lambda_fn = lambda x : x**2 # lambda function is defined
print(lambda_fn(3))  # we use the lambda function from inside the print statement.
```

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

Answer –

```
lst =[1,2,3,4,5,6] # a sample list
max_value = lambda lst: max(lst)
print(max_value(lst)) # we call the lambda function from inside the print statement.
```

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

Answer –

```
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]
```

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

<u>Answer</u> –

```
test_list = ["this","is","a","test","list"]
srt_list = list(sorted(test_list, key = lambda x:len(x), reverse=False))
print(srt_list)
```

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

<u>Answer</u> –

```
test_list1 = ["this","is","a","test","list"]
test_list2 = ["test","list"]
srt_list = list(filter(lambda x: x in test_list1,test_list2))
print(srt_list)
```

<u>Q 11.</u> Write a recursive function to calculate the factorial of a given positive integer.

<u>Answer</u> –

```
def fact(num):
    if num == 0:
        return 1

    else:

        return num * fact(num - 1)



    number = 6

    factorial = fact(number)

    print(f"The factorial of {number} is {factorial}")
```

<u>Q 12.</u> Implement a recursive function to compute the nth Fibonacci number.

<u>Answer</u> –

```
def fib(num):
    if num <= 0:
        return None
    elif num == 1:
        return 0
    elif num == 2:
        return 1
    else:
        return fib(num - 1) + fib(num - 2)



number = 7
fibonacci = fib(number)
print(f"The {number}th Fibonacci number is {fibonacci}")
```

<u>Q 13.</u> Create a recursive function to find the sum of all the elements in a given list.

<u>Answer</u> –

```
test_list = [1,2,3,4,5,6]

sum=0
for a in test_list:
    sum = sum + a
```

```
print(sum)
```

<u>Q 14.</u> Write a recursive function to determine whether a given string is a palindrome.

<u>Answer</u> –

```
test_string = "akhil"

rev =(test_string[::-1])

if test_string==rev:

   print("palindrome")

else:

   print("Not palindrome")
```

<u>Q 15.</u> Implement a recursive function to find the greatest common divisor (GCD) of two positive integers.

<u>Answer</u> –

```
def gcd(a, b):

   if b == 0:

       return a



   return gcd(b, a % b)



num1 = 24

num2 = 36

print(gcd(num1, num2))  # Output: 12
```