Assignment 9

11th June

Q1. What is a lambda function in Python, and how does it differ from a regular function?

Ans: A lambda function in Python is a small anonymous function that can take any number of arguments but can only have one expression. The syntax for a lambda function is 'lambda arguments : expression'. The expression is executed and the result is returned.

Lambda functions are defined on a single line using the 'lambda' keyword, while regular functions are defined using the 'def' keyword followed by the function name and any arguments. A lambda function is an expression which evaluates to a function object, while a 'def' statement has no value, and creates a function object and binds it to a name.

```
#Example

add = lambda x, y: x + y
print(add(5, 3))  # Output: 8
```

```
8
```

Q2. 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 have multiple arguments.

```
# Define a lambda function that takes three arguments and returns their product
product_num = lambda x, y, z: x * y * z

# Use the lambda function
result = product_num(5, 3, 8)
print(result)
```

```
120
```

Q3. How are lambda functions typically used in Python? Provide an example use case.

Ans: Lambda functions in Python are typically used as a convenient way to create small, anonymous functions. They are particularly useful in situations where you need a simple function for a short period of time and don't want to define a full-fledged named function.

```
#Example

#Printing thrice the of numbers provided
num = (3,5,6,7,8)
triple_num = list(map(lambda x: x*3, num))
print(triple_num)
```

```
[9, 15, 18, 21, 24]
```

Q4. What are the advantages and limitations of lambda functions compared to regular functions in Python?

Ans: The advantages lambda functions in Python include:

Concise and clear code: Lambda functions allow you to write concise and clear code, as they can be defined in a single line of code.

No additional variables: Since lambda functions do not require you to define a separate function, they do not add any additional variables to your code.

Can be passed immediately: Lambda functions can be passed immediately as arguments to other functions, without the need for additional variables or function definitions.

Automatic return of results: Lambda functions automatically return the result of their expression, without the need for an explicit return statement.

Useful for functional programming: Lambda functions are very useful in the context of functional programming, as they can be used with interfaces such as filter, map, and others.

Limitations of lambda functions in Python:

Limited to a single expression: Lambda functions can only contain one expression, meaning that they cannot contain multiple statements or complex control flow.

Difficult to read and understand: Since lambda functions can only be defined in a single line of code, they can be difficult to read and understand (especially by beginners).

Lack of names and documentation: Lambda functions lack names and documentation, meaning that the only way to know what they do is to read the code.

Cannot be used with complex functions: Lambda functions cannot be used with complex functions that require multiple arguments or statements.

Less flexible than named functions: Lambda functions cannot be referenced by name and can only be invoked when they are defined, which makes them less flexible than named functions.

Q5. 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 are able to access variables defined outside of their own scope. This is because lambda functions, like all functions in Python, have access to the variables in the enclosing scope.

```
#Example

x = 10
y = 20

add = lambda a, b: a + b + x + y

result = add(1, 2)
print(result) # 33
```

```
33
```

Q6. Write a lambda function to calculate the square of a given number.

```
square = lambda x: x**2
result = square(float(input("Enter number: ")))
print(result)
```

```
Enter number: 5
25.0
```

Q7. Create a lambda function to find the maximum value in a list of integers.

```
max_val = lambda lst: max(lst)
nums = [12,3,8,16,5,20]
result = max_val(nums)
print(result)
```

```
20
```

Q8. Implement a lambda function to filter out all the even numbers from a list of integers.

```
filter_even = lambda lst: list(filter(lambda x: x % 2 == 0, lst))
numbers = [3,12,9,6,8,5]
result = filter_even(numbers)
print(result)
```

```
[12, 6, 8]
```

Q9. Write a lambda function to sort a list of strings in ascending order based on the length of each string.

```
len_sort = lambda lst: sorted(lst, key=lambda s: len(s))
strings = ["Apple", "Banana", "Avocado", "Kiwi", "Blueberry"]
result = len_sort(strings)
print(result)
```

```
['Kiwi', 'Apple', 'Banana', 'Avocado', 'Blueberry']
```

Q10. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.

```
find_common = lambda list1, list2: list(filter(lambda x: x in list1,
list2))
list1 = [1, 2, 3, 4, 5, 6]
list2 = [4, 5, 6, 7, 8, 9]
result = find_common(list1, list2)
print(result)
```

```
[4, 5, 6]
```

Q11. Write a recursive function to calculate the factorial of a given positive integer.

```
def factorial(n):
    # Base case
    if n == 1:
        return 1
    # Recursive case
    else:
        return n * factorial(n-1)

result = factorial(int(input("Enter num: ")))
print(result)
```

```
Enter num: 6
720
```

Q12. Implement a recursive function to compute the nth Fibonacci number.

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

n = int(input("Enter num: "))
result = fibonacci(n)
print(f"The {n}th fibonacci number is ", result)
```

```
Enter num: 6
The 6th fibonacci number is  8
```

Q13. Create a recursive function to find the sum of all the elements in a given list.

```
def list_sum(lst):
    if not lst:
        return 0
    else:
        return lst[0] + list_sum(lst[1:])
my_list = [3,5,6,9,10]
result = list_sum(my_list)
print(f"The sum of the elements is: {result}")
```

```
The sum of the elements is: 33
```

Q14. Write a recursive function to determine whether a given string is a palindrome.

```
def is_palindrome(string):
    if len(string) <= 1:
        return True
```

```
    else:
        if string[0] == string[-1]:
            return is_palindrome(string[1:-1])
        else:
            return False

string1 = "radar"
string2 = "hello"
result1 = is_palindrome(string1)
result2 = is_palindrome(string2)
print(result1)
print(result2)
```

```
True
False
```

Q15. Implement a recursive function to find the greatest common divisor (GCD) of two positive integers.

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

result = gcd(24, 36)
print("The gcd of given numbers is: ", result)
```

```
The gcd of given numbers is:  12
```