Python Functions: Recursive, Lambda, Filter, Map, and Reduce

1. Recursive Functions in Python

Python mein **recursive function** wo function hota hai jo apne aap ko call karta hai. Ismein har call ke sath problem ka size chhota hota jata hai, jab tak wo **base case** tak nahi pohnch jata. Base case wo condition hoti hai jahan recursion ruk jata hai.

Syntax:

```
def function_name(parameters):
    if base_case_condition:
        return base_case_value
    else:
        return function_name(smaller_problem)
```

Example: Factorial Calculation

Factorial ek number ka wo product hota hai jo us number se le kar 1 tak hota hai. Jaise 5 ka factorial (5!) 5 * 4 * 3 * 2 * 1 hota hai.

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

print(factorial(5))  # Output: 120
```

Explanation:

- Humne `factorial()` function banaya jo recursive hai.
- Jab $n = 1$, to function 1 return karta hai (base case).
- Agar $n > 1$, to function apne aap ko call karta hai `n * factorial(n-1)`.
- Ye process tab tak chalegi jab tak n 1 nahi ho jata.

5 Additional Examples:

1. Fibonacci Sequence
Fibonacci series mein har number apne pehle do numbers ka sum hota hai.

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

print(fibonacci(5))  # Output: 5
```

2. Sum of Digits
Kisi number ke digits ka sum calculate karne ke liye recursion use karte hain.

```
def sum_digits(n):
    if n == 0:
        return 0
    else:
        return n % 10 + sum_digits(n // 10)

print(sum_digits(123))  # Output: 6
```

3. Power Function
Kisi number ko power mein raise karte hain.

```
def power(base, exp):
    if exp == 0:
        return 1
    else:
        return base * power(base, exp - 1)

print(power(2, 3))  # Output: 8
```

4. GCD (Greatest Common Divisor)
Do numbers ka GCD (sabse bara common divisor) find karna.

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

print(gcd(56, 98))  # Output: 14
```

5. Reverse a String
String ko reverse karna.

```
def reverse_string(s):
    if len(s) == 0:
        return s
    else:
        return reverse_string(s[1:]) + s[0]

print(reverse_string("hello"))  # Output: "olleh"
```

2. Lambda Functions in Python

Lambda function ek chhota sa anonymous function hota hai jo `lambda` keyword se define hota hai. Lambda functions koi name nahi rakhte aur sirf ek expression hota hai.

Syntax:

```
lambda arguments: expression
```

Example:

```
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

Explanation:

- Yaha `lambda` function do arguments `x` aur `y` leta hai, aur unka sum return karta hai.

15 Examples:

1. Multiplying two numbers:

```
multiply = lambda x, y: x * y
print(multiply(2, 3))  # Output: 6
```

2. Checking if a number is even:

```
is_even = lambda x: x % 2 == 0
print(is_even(4))  # Output: True
```

3. Calculating the square of a number:

```
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

4. Subtracting two numbers:

```
subtract = lambda x, y: x - y
print(subtract(7, 3))  # Output: 4
```

5. Returning the maximum of two numbers:

```
maximum = lambda x, y: x if x > y else y
print(maximum(10, 5))  # Output: 10
```

6. Returning the minimum of two numbers:

```
minimum = lambda x, y: x if x < y else y
print(minimum(10, 15))  # Output: 10
```

7. Calculating the absolute value of a number:

```
absolute_value = lambda x: x if x >= 0 else -x
print(absolute_value(-5))  # Output: 5
```

8. **String concatenation:**

```
concat = lambda x, y: x + y
print(concat("Hello, ", "World!"))  # Output: "Hello,
    World!"
```

9. **Squaring a number if it's positive:**

```
square_positive = lambda x: x ** 2 if x > 0 else "
    Invalid"
print(square_positive(4))  # Output: 16
```

10. **Convert Fahrenheit to Celsius:**

```
fahrenheit_to_celsius = lambda f: (f - 32) * 5/9
print(fahrenheit_to_celsius(32))  # Output: 0
```

11. **Checking if a string is a palindrome:**

```
is_palindrome = lambda s: s == s[::-1]
print(is_palindrome("madam"))  # Output: True
```

12. **Calculating the length of a string:**

```
length = lambda s: len(s)
print(length("Python"))  # Output: 6
```

13. **Rounding a number:**

```
round_number = lambda x: round(x, 2)
print(round_number(3.14159))  # Output: 3.14
```

14. **Formatting a string in uppercase:**

```
to_upper = lambda s: s.upper()
print(to_upper("hello"))  # Output: "HELLO"
```

15. **Finding the remainder of division:**

```
remainder = lambda x, y: x % y
print(remainder(10, 3))  # Output: 1
```

3. Lambda with Filter

`filter()` function ek iterable ko filter karta hai, jo lambda function ke given condition par based hota hai. Ye un elements ko return karta hai jo condition satisfy karte hain.

Syntax:

```
filter(function, iterable)
```

Example:

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(lambda x: x % 2 == 0,
   numbers))
print(even_numbers)  # Output: [2, 4, 6]
```

Explanation:

- `filter()` function lambda function ko use karta hai jisme condition `(x % 2 == 0)` hai, jo sirf even numbers ko filter karega.

5 Examples:

1. Filter positive numbers:

```
numbers = [-5, 4, 6, -3, 2]
positive = list(filter(lambda x: x > 0, numbers))
print(positive)  # Output: [4, 6, 2]
```

2. Filter even numbers:

```
numbers = [1, 2, 3, 4, 5, 6]
even = list(filter(lambda x: x % 2 == 0, numbers))
print(even)  # Output: [2, 4, 6]
```

3. Filter strings with more than 3 characters:

```
words = ["cat", "elephant", "dog", "fish"]
long_words = list(filter(lambda x: len(x) > 3, words))
print(long_words)  # Output: ['elephant']
```

4. Filter numbers divisible by 5:

```
numbers = [10, 15, 23, 30, 42]
divisible_by_5 = list(filter(lambda x: x % 5 == 0,
   numbers))
print(divisible_by_5)  # Output: [10, 15, 30]
```

5. Filter non-empty strings:

```
words = ["", "apple", "", "banana"]
non_empty = list(filter(lambda x: x != "", words))
print(non_empty)  # Output: ['apple', 'banana']
```

4. Lambda with Map

`map()` function kisi given function ko sequence ke har element par apply karta hai aur uska result return karta hai.

Syntax:

```
map(function, iterable)
```

Example:

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

Explanation:

- `map()` function har element par lambda function ko apply karta hai aur result return karta hai.

5 Examples:

1. Double the numbers:

```
numbers = [1, 2, 3, 4]
doubled = list(map(lambda x: x * 2, numbers))
print(doubled)  # Output: [2, 4, 6, 8]
```

2. Convert strings to uppercase:

```
words = ["hello", "world"]
upper_case = list(map(lambda x: x.upper(), words))
print(upper_case)  # Output: ['HELLO', 'WORLD']
```

3. Add 10 to each number:

```
numbers = [1, 2, 3, 4]
added_ten = list(map(lambda x: x + 10, numbers))
print(added_ten)  # Output: [11, 12, 13, 14]
```

4. Convert temperatures from Celsius to Fahrenheit:

```
celsius = [0, 25, 100]
fahrenheit = list(map(lambda c: (c * 9/5) + 32,
   celsius))
print(fahrenheit)  # Output: [32.0, 77.0, 212.0]
```

5. Square the numbers:

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

5. Lambda with Reduce

`reduce()` function ek rolling computation apply karta hai iterable ke har pair par. Ye function `functools` module se hota hai.

Syntax:

```
from functools import reduce
reduce(function, iterable)
```

Example:

```
from functools import reduce
numbers = [1, 2, 3, 4]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 24
```

Explanation:

- `reduce()` function multiplication operation apply kar raha hai har pair par.

5 Examples:

1. Sum of all numbers:

```
numbers = [1, 2, 3, 4]
total_sum = reduce(lambda x, y: x + y, numbers)
print(total_sum)  # Output: 10
```

2. Finding the maximum number:

```
numbers = [1, 5, 3, 4]
maximum = reduce(lambda x, y: x if x > y else y,
    numbers)
print(maximum)  # Output: 5
```

3. Concatenating strings:

```
words = ["hello", "world"]
concatenated = reduce(lambda x, y: x + "␣" + y, words)
print(concatenated)  # Output: "hello world"
```

4. Finding the minimum number:

```
numbers = [1, 5, 3, 4]
minimum = reduce(lambda x, y: x if x < y else y,
    numbers)
print(minimum)  # Output: 1
```

5. Multiplying all numbers:

```
numbers = [2, 3, 4]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 24
```