1. Lambda Expressions

Lambda expressions in Python are used for creating small anonymous functions.

- Example 1: lambda x, y: x + y

  Purpose: Adds two numbers x and y. This lambda expression is used to perform addition.

- Example 2: lambda x: x % 2 == 0

  Purpose: Checks if the number x is even. It returns True if the number is divisible by 2.

- Example 3: lambda a, b: a * b

  Purpose: Multiplies two numbers a and b. It is passed as a parameter to the apply_operation function to perform multiplication.

2. Functional Interfaces

In Python, functions themselves are first-class citizens, and we don’t use specific "functional interfaces" as in Java. Instead, we use functions directly.

- Example 1: filter(lambda x: x % 2 == 0, numbers)

  Purpose: The filter function takes a function (lambda) and a collection (numbers) and applies the function to filter out even numbers.

- Example 2: map(math.sqrt, numbers)

  Purpose: The map function applies the math.sqrt function to each element of the numbers list to compute the square roots.

3. Streams (or Iterable Operations)

While Python does not have a Stream API like Java, it does support functional-style operations through iterators and generators.

- Example 1: list(filter(lambda x: x % 2 == 0, numbers))

  Operations:

  filter(lambda x: x % 2 == 0, numbers) filters even numbers from the list.
  list() converts the filtered results into a list.

  Role: Functional approach for filtering data.
- Example 2: list(map(math.sqrt, numbers))

Operations:

map(math.sqrt, numbers) applies the sqrt function to each element of the list.

list() converts the mapped results into a list.

Role: Transforms the data by applying a function to each element.

4. Method References

In Python, method references are usually implemented by directly using functions, but Python allows passing methods as arguments.

- Example: math.sqrt

  Usage: This is used as a method reference to apply sqrt to each element of the list through map.
- Example: print

  Usage: The built-in print method is passed as a callback function to output the results in the generator loop.

5. Immutable Data Structures

No immutable Data Structure in the give code
Python allows the creation of immutable data structures through tuples and frozensets. However, lists in the code are mutable. Here's an example of an immutable structure:

- Example: numbers = [1, 2, 3, 4, 5, 6] (Mutable)
  T = (1,2,3,4,5,6) (Immutable Tuple)

  Significance: Lists are mutable, but we could use tuple(numbers) for an immutable version.
- Significance: Immutable data structures prevent side effects and promote functional programming purity.

6. Higher-Order Functions

Higher-order functions in Python are functions that take other functions as arguments or return functions.

- Example: apply_operation(6, 7, lambda a, b: a * b)

  Purpose: Accepts a function (lambda a, b: a * b) as an argument and applies it to 6 and 7. The function performs multiplication.
- Role: This allows dynamic function selection, providing flexibility in performing operations.

7. Recursion

Recursion occurs when a function calls itself to solve smaller instances of a problem. The factorial function in the Python code is a recursive function.

- Example: factorial(5)

  Functionality: Computes the factorial of a number by recursively calling itself with a smaller value of n until the base case (n == 0 or n == 1) is reached.
- Role: Recursion is used here instead of an iterative loop to calculate factorial, following functional programming principles.

8. Lazy Evaluation

Lazy evaluation delays the computation of values until they are explicitly required. Python's generators are an example of lazy evaluation.

- Example: (random.random() for _ in range(5))

  Description: This creates a generator that produces random numbers only when they are requested, i.e., lazily evaluated.
- Impact: Lazy evaluation ensures that random numbers are generated only when the generator is iterated over, saving memory and computation resources.