**Map, Filter, and Reduce in Python**

Python provides several built-in functions that are essential to functional programming: map(), filter(), and reduce(). These functions enable concise and expressive operations on collections.

**map()**

**Definition**: The map() function applies a given function to all items in an input list (or any other iterable) and returns an iterator of the results.

**Syntax**:

map(function, iterable, ...)

* function: A function that takes one or more arguments.
* iterable: One or more iterables (lists, tuples, etc.).

**Example**:

def square(x):

return x \* x

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

squared\_numbers = map(square, numbers)

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

**Using lambda with map():**

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

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

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

**Example with multiple iterables**:

a = [1, 2, 3]

b = [4, 5, 6]

summed = map(lambda x, y: x + y, a, b)

print(list(summed)) # Output: [5, 7, 9]

**filter()**

**Definition**: The filter() function constructs an iterator from elements of an iterable for which a function returns true.

**Syntax**:

filter(function, iterable)

* function: A function that returns a Boolean value.
* iterable: An iterable (list, tuple, etc.).

**Example**:

def is\_even(x):

return x % 2 == 0

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

even\_numbers = filter(is\_even, numbers)

print(list(even\_numbers)) # Output: [2, 4]

**Using lambda with filter():**

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

even\_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even\_numbers)) # Output: [2, 4]

**reduce()**

**Definition**: The reduce() function applies a function of two arguments cumulatively to the items of an iterable, from left to right, to reduce the iterable to a single value. reduce() is part of the functools module.

**Syntax**:

functools.reduce(function, iterable[, initializer])

* function: A function that takes two arguments.
* iterable: An iterable (list, tuple, etc.).
* initializer (optional): A value to start the reduction.

**Example**:

from functools import reduce

def add(x, y):

return x + y

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

sum\_of\_numbers = reduce(add, numbers)

print(sum\_of\_numbers) # Output: 15

**Using lambda with reduce():**

from functools import reduce

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

sum\_of\_numbers = reduce(lambda x, y: x + y, numbers)

print(sum\_of\_numbers) # Output: 15

**Example with an initializer**:

from functools import reduce

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

sum\_of\_numbers = reduce(lambda x, y: x + y, numbers, 10)

print(sum\_of\_numbers) # Output: 25 (10 + 1 + 2 + 3 + 4 + 5)

**Practical Examples**

**Example with map(), filter(), and reduce()**

Suppose you have a list of numbers, and you want to find the sum of the squares of the even numbers.

from functools import reduce

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

# Step 1: Filter even numbers

even\_numbers = filter(lambda x: x % 2 == 0, numbers)

# Step 2: Square the even numbers

squared\_even\_numbers = map(lambda x: x \* x, even\_numbers)

# Step 3: Sum the squared even numbers

sum\_of\_squares = reduce(lambda x, y: x + y, squared\_even\_numbers)

print(sum\_of\_squares) # Output: 20 (2^2 + 4^2)

**Summary**

* **map()**: Applies a function to all items in an iterable and returns an iterator of the results.
* **filter()**: Constructs an iterator from elements of an iterable for which a function returns true.
* **reduce()**: Applies a function cumulatively to the items of an iterable, reducing it to a single value.

These functions are powerful tools for functional programming in Python, enabling concise and expressive data transformations.