**WEEK-1**

**Functional Programming and Data Parsing**

**DESCRIPTION**

This unit focuses on advanced core Python concepts, primarily revolving around functional programming and data parsing techniques using JSON, XML, and regular expressions.

Functional programming in Python emphasizes the use of functions as first-class citizens. This approach promotes code reusability, clarity, and brevity. Key concepts include:

* **Lambda Functions**: Anonymous functions defined with the lambda keyword. Useful for short operations, especially when used in conjunction with higher-order functions.
* **Map**: Applies a function to all the items in an input list (or any iterable).
* **Filter**: Filters elements of an iterable based on a function that returns either True or False.
* **Reduce**: Performs a rolling computation to sequential pairs of elements. Requires the functools module.
* **Iterators**: Objects that can be iterated upon and implement \_\_iter\_\_() and \_\_next\_\_() methods.
* **Generators**: A simpler way to create iterators using functions and the yield keyword. They are memory efficient as values are produced on the fly.
* **List Comprehensions**: A concise way to create lists using a single line of code with conditions and loops embedded.

The unit also introduces **parsing and handling data formats** commonly used in real-world applications:

* **JSON (JavaScript Object Notation)**: A lightweight data-interchange format. Python’s json module allows conversion between JSON and Python dictionaries using json.dumps(), json.loads(), etc.
* **XML (eXtensible Markup Language)**: Python’s xml.etree.ElementTree provides tools to parse and manipulate XML data.
* **Regular Expressions**: Powerful for pattern matching and text manipulation using Python’s re module. They help in tasks like validating email formats, extracting dates, phone numbers, and more.

By the end of this unit, students will be able to write concise functional code, process real-world data formats like JSON and XML, and manipulate text using regular expressions.

**1.Write a program to calculate the square of even numbers in a list and sum them using map, filter, and reduce.**

**Description:**

This experiment demonstrates the use of functional programming techniques in Python. The program filters even numbers from a list, maps them to their squares, and then reduces them to a single sum. It shows the effective combination of filter(), map(), and reduce() functions for concise and readable code.

**Program** :

from functools import reduce

# List of numbers

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filter even numbers

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

# Square the even numbers

squared = list(map(lambda x: x\*\*2, even\_numbers))

# Sum the squared values

result = reduce(lambda x, y: x + y, squared)

print("Even Numbers:", even\_numbers)

print("Squared Even Numbers:", squared)

print("Sum of Squares of Even Numbers:", result

**Output:**

****

**VIVA QUESTIONS**

**UNIT – 1: Viva Questions with Answers**

1. **What is the difference between map(), filter(), and reduce()?**

	* map() applies a function to every item in an iterable.
	* filter() returns items for which the function returns True.
	* reduce() applies a function cumulatively to the items and returns a single result. (Needs functools.)
2. **How is a lambda function different from a regular function?**

A lambda function is anonymous and defined using the lambda keyword. It contains only a single expression, unlike regular functions defined with def, which can have multiple statements.
3. **What is a generator? How is it different from a normal function?**

A generator is a function that uses the yield keyword to return values one at a time, pausing between each. Unlike normal functions that return once, generators maintain state between yields and are more memory efficient.
4. **Write a Python list comprehension to generate a list of squares of even numbers from 0 to 10.**

squares = [x\*\*2 for x in range(11) if x % 2 == 0]