**WEEK-2**

**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.

**Write a program to save data into a JSON and XML file, read it back, and modify it.**

**Description:**

This experiment helps understand how to work with real-world data formats like JSON and XML. The student learns how to write data to a file, read it back, and perform modifications. It introduces Python's json module for JSON handling and xml.etree.ElementTree for XML manipulation.

**Program :**

import json

# Create and write data

data = {

 "name": "John",

 "age": 30,

 "skills": ["Python", "JavaScript"]

}

with open("user.json", "w") as f:

 json.dump(data, f, indent=4)

# Read data

with open("user.json", "r") as f:

 user = json.load(f)

 print("Before Update:", user)

# Update data

user["age"] = 31

user["skills"].append("SQL")

with open("user.json", "w") as f:

 json.dump(user, f, indent=4)

# Delete key

del user["skills"]

with open("user.json", "w") as f:

 json.dump(user, f, indent=4)

print("After Delete:", user)

**Output:**

****

import xml.etree.ElementTree as ET

# Create XML structure

person = ET.Element("person")

ET.SubElement(person, "name").text = "John"

ET.SubElement(person, "age").text = "30"

ET.SubElement(person, "city").text = "London"

tree = ET.ElementTree(person)

tree.write("person.xml")

# Read and update XML

tree = ET.parse("person.xml")

root = tree.getroot()

# Update age

for elem in root:

 if elem.tag == "age":

 elem.text = "31"

tree.write("updated\_person.xml")

# Display updated content

for elem in root:

 print(f"{elem.tag}: {elem.text}")

**Output:**

`

**VIVA QUESTIONS**

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

1. **What is the use of json.dumps() and json.loads()?**

	* json.dumps() converts a Python object into a JSON string.
	* json.loads() converts a JSON string into a Python object (like a dictionary).
2. **How do you parse XML data in Python?**

You can use the xml.etree.ElementTree module to parse XML. Use ET.parse() to read from a file and ET.fromstring() to parse XML from a string.

1. **What are the key differences between JSON and XML?**

**Syntax:** JSON uses a lightweight, JavaScript-like syntax with key-value pairs, while XML uses nested tags.

**Readability:** JSON is more concise and human-readable.

**Data Types:** JSON supports data types like strings, numbers, booleans, and arrays; XML treats everything as text.

**Parsing:** JSON is faster to parse in most programming languages, including Python, using the json module; XML parsing uses xml.etree.ElementTree or other libraries.

**Use Case:** JSON is commonly used in web APIs; XML is often used in legacy systems and document storage.