Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

AIDS - 2 Lab
Experiment - 3

Aim: Parsing using Python

Roll No.	70
Name	MAYURI SHRIDATTA YERANDE
Class	D20B
Subject	AIDS - 2
Grade:	

EXPERIMENT - 3

AIM: Parsing using Python

THEORY:

Parsing in Python refers to the process of analyzing a piece of text, often in a specific format or structure, to extract relevant information or manipulate its components. Parsing is commonly used in various applications such as data extraction, language processing, configuration file reading, and more.

There are various types of parsing, and Python offers libraries and tools to facilitate parsing in different contexts.

Text Parsing:

Text parsing involves breaking down a plain text into meaningful components. Python's built-in string manipulation functions can often suffice for simple text parsing tasks. Regular expressions (using the re module) provide a more powerful tool for pattern matching and extraction.

This can be useful in a variety of scenarios, such as data extraction, text analysis, natural language processing, and more. The process often includes pattern matching, tokenization, and manipulation of the text's elements.

Applications:

1. **Resume/CV Parsing:**
   Resume parsing involves extracting structured information from resumes or CVs to populate fields in a database or application. This is commonly used in recruitment and job application systems.
   Example: A Python library like Pyresparser can be used to parse resumes and extract information such as contact details, skills, work experience, and education.

2. **Log File Analysis:**
   Parsing log files generated by software applications or systems can help identify errors, trends, and performance issues.
   Example: Using Python to parse and analyze web server logs to identify frequently accessed URLs or detect patterns of suspicious activity.

3. **Configuration File Parsing:**
   Many software applications use configuration files to store settings. Parsing these files enables programs to read and utilize configuration options.
   Example: Parsing an INI or YAML configuration file to extract settings for a software application.

4. **Web Scraping**:
   Web scraping involves parsing HTML content from web pages to extract specific data. It's used for data extraction, content aggregation, and more.
   Example: Using libraries like BeautifulSoup or Scrapy to parse web pages and extract information like product prices, news headlines, or weather data.

5. **Language Processing and NLP:**
   Parsing is fundamental in natural language processing tasks, where text is analyzed for meaning and structure.
   Example: Using the NLTK library to parse sentences and identify grammatical structures or using spaCy to perform part-of-speech tagging and named entity recognition.

Advantages of Parsing Using Python:

1. **Rich Ecosystem:** Python has a wide range of parsing libraries and tools available, making it versatile for parsing various types of data formats, such as XML, JSON, HTML, CSV, etc.

2. **Easy-to-Learn:** Python's syntax is relatively simple and easy to understand, which makes it accessible for beginners to learn and work with parsing techniques.

3. **Built-in Libraries:** Python comes with built-in libraries like re (regular expressions) that allow you to perform text parsing efficiently for simple cases without requiring external dependencies.

4. **Third-party Libraries:** Python has powerful third-party parsing libraries like BeautifulSoup (for HTML/XML), json (for JSON data), csv (for CSV files), and more. These libraries simplify the parsing process by providing convenient APIs.

5. **Regular Expressions:** Python's re module allows you to work with regular expressions, enabling complex pattern matching and extraction from strings. This is particularly useful for parsing data with specific patterns.

6. **Data Transformation:** Python's parsing capabilities enable you to convert data from one format to another, facilitating data integration and manipulation tasks.

7. **Community Support**: Python has a large and active community of developers. If you encounter any issues or need assistance with parsing, you can find a wealth of resources online.

8. **Cross-Platform:** Python is a cross-platform language, meaning you can perform parsing tasks on different operating systems without major compatibility issues.

Disadvantages of Parsing Using Python:

1. **Performance:** While Python is relatively easy to use, it may not be the most performant option for very large or complex parsing tasks. Parsing can be resource-intensive, and Python's interpreted nature might lead to slower execution compared to compiled languages.

2. **Dependency Management:** Third-party libraries are often used for advanced parsing tasks. Managing dependencies and ensuring compatibility between different libraries can sometimes be challenging.

3. **Error Handling**: Error handling during parsing can be complex, especially when dealing with malformed or inconsistent data. Ensuring robust error handling can add complexity to your parsing code.

4. **Learning Curve for Advanced Techniques:** While basic parsing is straightforward, mastering advanced parsing techniques (e.g., parsing binary formats) or using more complex libraries can require a steep learning curve.

5. **Security Concerns:** Poorly handled parsing can lead to security vulnerabilities like code injection or data leakage. It's important to validate and sanitize input data to prevent these issues.

6. **Fragmented Approach**: Different data formats often require different parsing libraries or techniques. This can lead to a fragmented approach where you need to learn and use multiple methods for different tasks.

7. **Limited Context Awareness:** Python's built-in parsing capabilities might not provide sophisticated context awareness, which could be important for parsing highly complex structures.

EXPLANATION:-

- Importing Required Modules
- A list named RESERVED_WORDS contains specific keywords that are expected to indicate the start of different sections in the resume.
- The function uses the docx2txt module to extract the text content from a DOCX file given its docx_path. If text is successfully extracted, it returns the extracted text; otherwise, it returns None.
- Next function takes the extracted text as input_text and processes it line by line. It iterates through each line and checks if any of the RESERVED_WORDS is present in the line (case-insensitive).
- If found, it sets the current_section to the matched word and initializes an empty list for that section in the education_sections dictionary. Then, if there's a current_section and the line is not empty after stripping, the line's content is added to the corresponding section in the dictionary.
- The main block of code executes only if the script is run as the main program. It extracts the text from the DOCX file then uses the extract_education_sections function to extract specific sections related to education. Finally, it prints out each section's title and its content, separated by a line of dashes.

IMPLEMENTATION:

Code:
```
import docx2txt
import re

RESERVED_WORDS = [
    'Objective',
    'Academic Details',
    'Skills',
    'Experiences',
    'Projects',
    'Honors',
    'Achievements',
]

def extract_sections_from_docx(docx_path):
    txt = docx2txt.process(docx_path)
    if txt:
        return txt
    return None

def extract_education_sections(input_text):
    education_sections = {}
    current_section = None

    for line in input_text.split('\n'):
        for word in RESERVED_WORDS:
            if word.lower() in line.lower():
                current_section = word
                education_sections[current_section] = []
                break

        if current_section and line.strip():

education_sections[current_section].append(line.strip())

    return education_sections

if __name__ == '__main__':
    text = extract_sections_from_docx('Aaman_Resume_Doc.docx')
    education_sections = extract_education_sections(text)

    for section, content in education_sections.items():
        print(section)
        print('\n'.join(content))
        print('-' * 40)
```

Output:

```
Objective
Objective:                                                          .
I have a passion for finding effective solutions to difficult challenges. I have a strong work ethic, am constantly honing my craft, and produce excellent work. I'm open to learning about new technology, and I appreciate working with people to come up with creative solutions
----------------------------------------
Experiences
Experiences:                                                        .
Pixolo Production
FrontEnd Developer Intern | January 2023 - February 2023 (2 months)
Experienced in employing diverse frontend frameworks for website development.
```

Engage in team collaboration, actively contributing to project architecture and continuous improvement efforts.
IEEE VESIT
Junior Web Developer | August 2022 - April 2023 (9 months)
I have successfully contributed to Web development, showcasing meticulous attention to detail & a strong grasp of editorial practices.

Skills
Skills: .
Programming Languages: Java, Python, JavaScript
Framework: Flask, Django, MERN Stack, React Native (Ionic)
Databases: MySQL, SQLite, FireBase, MongoDB
Frontend Technologies: HTML, CSS, JS, SCSS/SASS, React JS, Bootstrap, Figma, Canva
DevOps: Git, GitHub
Tools: Tableau, Power BI

Academic Details
Academic Details: .
Qualification
Institute/Organization
Percentage / CGPA
Year
B.E in INFT
V. E. S. I. T., Chembur
9.63 (Agg. till V Sem)
2020 - 2024 (Present)
HSC
V. G. Vaze College, Mulund
81.54 %
March 2020
SSC
Vani Vidyalaya, Mulund
86.40 %
March 2018

Projects
Projects: .
MediOlx (Reselling unused Medicines & Medical Equipment)
Flask, HTML, CSS, JS, Bootstrap, MySQL

Designed a platform to resell the unused Medicine (not Expired) & Medical Equipment to reduce the medicinal waste and to make them available to those who need it, with a discounted price.
DemandWise (Inventory Demand Forecasting)
AI/ML, Python, Flask, HTML, CSS, JS, Bootstrap
Built a platform where users can upload past sales data for products they want to forecast. Using machine learning, the platform predicts the required product quantities for the next 60 days.
EduAbled (Edtech Platform for disabled kids)
MERN Stack, Bootstrap, Firebase
Designed a platform focused for children, having difficulties in reading and writing. We integrated Voice navigations and PDF reader, also with video lectures and study materials to help them in studies.
BankWiz (Banking Dashboard for Corporates)
Next.js, Tailwind, Django, PostgreSQL, Tableau
Systemized a comprehensive banking dashboard for corporates, providing visualization of all bank data, including transaction history, foreign exchange and investments.
Stocks Trend Prediction
Python, Streamlit
Stock trend prediction using LSTM is a deep learning technique that uses historical price and volume data to predict future price movements. LSTM is well-suited for this task because it can capture long-term dependencies in time series data and can handle noisy and unpredictable data.

Honors
Honors and Achievements: .
Represented the college in the National Level Corporate Hackathon, SemiColon’23, organized by Persistent Co.
Winner of Syrus’23 Hackathon, Web 2.0 - 24 hours Hackathon (CodeCell-VESIT)
2nd Runner-up in LBS’23 - Looking Beyond Syllabus Hackathon (Hardware Section)
Winner of Brand-a-thon’23 (VESIT Renaissance Cell)
Winner of Web-a-thon’22 (CSI-VESIT)
Winner of INVICTUS’22 - 24 hours Hackathon (ISTE -VESIT x QuestIT-VESIT)

CONCLUSION: Parsing using Python provides a flexible and accessible approach to extracting meaningful information from various types of data formats, such as text, HTML, XML, JSON, and more. Python's extensive ecosystem of libraries and tools, combined with its easy-to-learn syntax, makes it a popular choice for parsing tasks. Thus we successfully performed parsing using python.