# Information Retrieval from PDF Documents

## Part 1: Narrate and Illustrate the Problem

### Problem:

#### Information Retrieval from PDF Documents

Many organizations store critical data in PDF documents. Retrieving specific information, like searching for keywords, extracting sentences, or identifying patterns, can be challenging when handling large PDFs. A user-friendly system is needed to:

1. Upload PDF files.
2. Extract all text from the PDF.
3. Search for specific keywords within the extracted text.
4. Display results in a user-friendly format.

**Illustration:** Suppose a company stores its employee profiles in PDFs. HR needs to identify all employees with skills related to "Web Builder." Our system automates this process by extracting text from the PDF and highlighting sentences containing the keyword.

## Part 2: Algorithm to Solve the Problem

### Algorithm:

1. **Input Validation:**
    - Accept a file path.
    - Verify if the file exists and is a valid PDF.
2. **Text Extraction:**
    - Use `PyPDF2` to read and extract text from the PDF.
3. **Text Splitting:**
    - Split the extracted text into sentences using a delimiter-based approach.
4. **Keyword Search:**
    - Iterate through each sentence.
    - For each keyword, check if it exists in the sentence (case-insensitive).
5. **Output Results:**

    - Group and display sentences for each keyword.

### Why This Algorithm is Suitable:

- **Efficiency:** Extracting text and searching within sentences minimizes unnecessary operations.
- **Scalability:** Supports large PDFs with multiple pages and multiple keywords.
- **Flexibility:** Works for a wide range of documents.

# Part 3: Code Implementation

The code demonstrates OOP principles such as encapsulation, modularity, and reusability. Here's the final working solution:

Answer : Check the Codes

Here's a comprehensive solution for your assignment:

---

# Part 1: Narrate and Illustrate the Problem

## Problem:

#### Information Retrieval from PDF Documents

Many organizations store critical data in PDF documents. Retrieving specific information, like searching for keywords, extracting sentences, or identifying patterns, can be challenging when handling large PDFs. A user-friendly system is needed to:

1. Upload PDF files.
2. Extract all text from the PDF.
3. Search for specific keywords within the extracted text.
4. Display results in a user-friendly format.

**Illustration:** Suppose a company stores its employee profiles in PDFs. HR needs to identify all employees with skills related to "Web Builder." Our system automates this process by extracting text from the PDF and highlighting sentences containing the keyword.

---

# Part 2: Algorithm to Solve the Problem

## Algorithm:

1. **Input Validation:**
    - Accept a file path.
    - Verify if the file exists and is a valid PDF.
2. **Text Extraction:**
    - Use `PyPDF2` to read and extract text from the PDF.
3. **Text Splitting:**
    - Split the extracted text into sentences using a delimiter-based approach.
4. **Keyword Search:**
    - Iterate through each sentence.
    - For each keyword, check if it exists in the sentence (case-insensitive).
5. **Output Results:**
    - Group and display sentences for each keyword.

## Why This Algorithm is Suitable:

- **Efficiency:** Extracting text and searching within sentences minimizes unnecessary operations.
- **Scalability:** Supports large PDFs with multiple pages and multiple keywords.
- **Flexibility:** Works for a wide range of documents.

---

# Part 3: Code Implementation

The code demonstrates OOP principles such as encapsulation, modularity, and reusability. Here's the final working solution:

---

# Part 4: Computational Complexity

## Time Complexity:

1. **PDF Validation: O(1)**
   **Text Extraction: O(n)**, where n is the number of pages in the PDF.
2. **Sentence Splitting: O(m)**, where m is the total number of characters in the text.
3. **Keyword Search: O(k · s)**, where k is the number of keywords and sss is the number of sentences.

*Overall Time Complexity:*
**O(n + m + k · s)**

### Space Complexity:

1. **Text Storage: O(m)**, where mmm is the total characters in the text.
2. **Search Results Storage: O(k · r)**, where rrr is the average number of results per keyword.

*Overall Space Complexity:*
**O (m + k · r)**

# Summary:

This solution efficiently extracts and searches text from PDFs, leveraging OOP principles and Python libraries. It balances simplicity with scalability, making it practical for real-world applications. Let me know if further refinements or explanations are needed!