🧠 What Does This Program Do?

It allows you to:

1. **Read a PDF file** (like your notes).
2. **Extract the text** from the PDF pages.
3. **Ask questions** about the content.
4. Use a **pre-trained AI model** to find answers inside the notes.

```
import pdfplumber
from transformers import pipeline

# Step 1: Extract text from PDF using pdfplumber
def extract_text_from_pdf(pdf_path):
    text = ""
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"
    return text

# Step 2: Split text into smaller chunks (to fit model input limits)
def split_text(text, max_words=1000):
    words = text.split()
    return [' '.join(words[i:i + max_words]) for i in range(0, len(words),
max_words)]

# Step 3: Find the best answer from all chunks
def find_best_answer(question, context_chunks, qa_pipeline):
    best_score = 0
    best_answer = "Sorry, I don't know."
    for chunk in context_chunks:
        result = qa_pipeline(question=question, context=chunk)
        if result['score'] > best_score:
            best_score = result['score']
            best_answer = result['answer']
    return best_answer

# Step 4: Main function to load PDF and answer questions
def main():
    pdf_path = r"F:\python demo\notes.pdf"  # Replace with your own PDF path
    print("Reading PDF...")
    context = extract_text_from_pdf(pdf_path)

    if not context.strip():
        print("⚠️ No text found in the PDF. Is it scanned or empty?")
        return

    print("Splitting content into chunks...")
    chunks = split_text(context)

    print("Loading AI model...")
    qa_pipeline = pipeline("question-answering", model="distilbert-base-cased-
distilled-squad")

    print("\nReady! Ask a question about your notes (type 'exit' to quit):")
    while True:
        question = input("\nAsk a question: ")
        if question.lower() == 'exit':
            print("Goodbye!")
            break
        answer = find_best_answer(question, chunks, qa_pipeline)
        print("Answer:", answer)

# Run the program
if __name__ == "__main__":
    main()
```

💡 Step-by-Step Explanation

🔷 Importing Required Libraries

```
python
CopyEdit
import pdfplumber
from transformers import pipeline
```

- `pdfplumber` helps extract **text from PDF files**. It's better than other libraries like `PyPDF2`, especially for formatted PDFs.
- `transformers` is from Hugging Face — it gives you access to **pre-trained AI models** (like BERT) for tasks like question answering.

🔷 Step 1: Extract Text from PDF

```
python
```

```
CopyEdit
def extract_text_from_pdf(pdf_path):
    text = ""
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"
    return text
```

What’s happening here:

- You give it a **PDF file path**.
- It **opens the PDF**, reads each **page one by one**.
- It tries to **get text from each page**.
- If there's text, it adds it to a big string called `text`.
- Finally, it returns all the text from your PDF.

📝 If your PDF is **scanned (image-based)** and not actual text, this won't work. You’d need OCR (Optical Character Recognition) like Tesseract.

🔹 Step 2: Split Text into Chunks

```
python
CopyEdit
def split_text(text, max_words=1000):
    words = text.split()
    return [' '.join(words[i:i + max_words]) for i in range(0, len(words),
max_words)]
```

Why do we need this?

- AI models (like BERT) have a **limit** on how much text they can handle at once (usually around 512–1024 words).
- So, this function **splits** your text into **smaller parts** (called chunks).

How does it work?

- It **splits the full text into words**.
- Then it **groups them into blocks** of 1000 words.
- Returns a **list of these blocks**.

🔹 Step 3: Find the Best Answer from All Chunks

```
python
CopyEdit
def find_best_answer(question, context_chunks, qa_pipeline):
    best_score = 0
    best_answer = "Sorry, I don't know."
    for chunk in context_chunks:
        result = qa_pipeline(question=question, context=chunk)
        if result['score'] > best_score:
            best_score = result['score']
            best_answer = result['answer']
    return best_answer
```

What this does:

- You ask a **question**.
- The AI **looks at each chunk** of your notes and tries to answer it.
- It **scores** each answer with a confidence value.
- It **keeps the best-scoring answer** and returns it.

💡 The higher the `score`, the more confident the model is.

🧩 Step 4: Main Program Logic

```
python
CopyEdit
def main():
    pdf_path = r"F:\python demo\notes.pdf"
    ...
```

This is the **main controller** of your program.

1. **Reads the PDF**

   ```
   python
   CopyEdit
   context = extract_text_from_pdf(pdf_path)
   ```

2. **Checks if any text was extracted**

   ```
   python
   CopyEdit
   if not context.strip():
       print("⚠ No text found in the PDF. Is it scanned or empty?")
       return
   ```

3. **Splits the text**

   ```
   python
   CopyEdit
   chunks = split_text(context)
   ```

4. **Loads the AI model**

   ```
   python
   CopyEdit
   qa_pipeline = pipeline("question-answering", model="distilbert-base-
   cased-distilled-squad")
   ```

This creates a **Q&A AI system** using a small but efficient model called **DistilBERT** trained on the SQuAD dataset.

5. **Keeps asking questions in a loop**

```
python
CopyEdit
while True:
    question = input("\nAsk a question: ")
    ...
    answer = find_best_answer(question, chunks, qa_pipeline)
    print("Answer:", answer)
```

You can keep asking questions until you type `"exit"`.

□ Example Run

```
vbnet
CopyEdit
Reading PDF...
Splitting content into chunks...
Loading AI model...

Ready! Ask a question about your notes (type 'exit' to quit):

Ask a question: What is a Python class?
Answer: A class is a blueprint for creating objects.
```

✅ Summary

Component	Purpose
`pdfplumber`	Reads and extracts text from PDFs
`pipeline()`	Loads a pre-trained AI model
`split_text()`	Splits long text into smaller parts
`find_best_answer()`	Searches all parts for the best answer
`main()`	Connects everything and handles user input