Prerequisites for the PDF Protection Tool Project

By: **Inlighn Tech**

Overview

The PDF Protection Tool project involves creating a Python script that adds password encryption to an existing PDF file, producing a secure output file. The provided solution uses the PyPDF2 library to read and write PDF content, encrypts it with a user-specified password, and handles errors and command-line arguments. To write this script independently, students must master several key concepts and skills from the curriculum. These prerequisites, supported by practical Python code examples, ensure students understand how PDF protection works and can implement the script effectively on their local machines.

Prerequisite Knowledge and Skills

1. **File Input/Output (I/O) Basics**
    - **What to Know**:
        - Reading and writing files in Python using open() in binary mode ('rb' for reading, 'wb' for writing).

        - Context managers (with statements) to safely handle file operations and ensure closure.
    - **How It Applies**:
        - The script reads an input PDF (input_pdf) in binary mode, processes it, and writes the encrypted result to output_pdf.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 11 (IO with Basic Files).
        - "WORKING WITH DATA STRUCTURES & FILE HANDLING" Lesson 10 (Opening and Reading Files and Folders Python OS Module).
    - **Practical Prep**:
        - Practice basic file I/O with binary data.
    - **Python Code Block**:

```
# Copy a file in binary mode
input_file = "test.txt"
output_file = "test_copy.txt"

with open(input_file, "wb") as f:
   f.write(b"Sample text")  # Write sample data
with open(input_file, "rb") as source:
   data = source.read()
   with open(output_file, "wb") as dest:
      dest.write(data)
print(f"Copied {input_file} to {output_file}")
```

    -

        - **Run It**: Save as file_io_test.py, run with python file_io_test.py. Creates and copies a file, mimicking the script's binary PDF handling.

2. **Functions and Arguments**
    - **What to Know**:
        - Define functions with def to organize code (e.g., cretae_password_protected_pdf).
        - Pass multiple arguments (e.g., input_pdf, output_pdf, password) to functions.
    - **How It Applies**:

        - The script uses a function to encapsulate PDF encryption logic, taking input file, output file, and password as parameters.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 17-19 (Introduction to Functions, def Keyword, Basics of Functions).
    - **Practical Prep**:
        - Experiment with function arguments.
    - **Python Code Block**:

```
# Function with multiple arguments
def combine_files(file1, file2, output):
    with open(file1, "r") as f1, open(file2, "r") as f2:
        content = f1.read() + f2.read()
        with open(output, "w") as out:
            out.write(content)
    print(f"Combined {file1} and {file2} into {output}")

combine_files("file1.txt", "file2.txt", "combined.txt")
```

    -
        - **Run It**: Create file1.txt and file2.txt with some text, save as function_test.py, run with python function_test.py. Shows passing arguments, similar to the script's function.

3. **Exception Handling**
    - **What to Know**:
        - Use try/except to manage errors (e.g., file not found, invalid PDF).
        - Handle specific exceptions like FileNotFoundError and generic ones with Exception.
    - **How It Applies**:
        - The script catches errors during file opening, PDF reading, and encryption to prevent crashes and provide feedback.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).
    - **Practical Prep**:
        - Test error handling with multiple exceptions.

- **Python Code Block**:

```
# Handle file-related errors
def read_file(filename):
    try:
        with open(filename, "r") as f:
            print(f.read())
    except FileNotFoundError:
        print(f"Error: {filename} not found")
    except Exception as e:
        print(f"Unexpected error: {e}")

read_file("nonexistent.txt")
```

-

    - **Run It**: Save as error_test.py, run with python error_test.py. Demonstrates catching file errors, preparing for the script's error handling.

4. **Command-Line Arguments with sys**
    - **What to Know**:
        - The sys module provides sys.argv to access command-line arguments passed to a script.
        - Validate argument count and extract values (e.g., input_pdf = sys.argv[1]).
    - **How It Applies**:
        - The script uses sys.argv to get input_pdf, output_pdf, and password from the command line.
    - **Curriculum Source**:
        - Not explicitly listed in your curriculum; assumes basic Python knowledge or a brief intro.
    - **Practical Prep**:
        - Practice parsing command-line inputs.
    - **Python Code Block**:

```
# Parse command-line arguments
import sys
```

```
if len(sys.argv) != 3:
    print("Usage: python script.py <name> <age>")
    sys.exit(1)

name = sys.argv[1]
age = sys.argv[2]
print(f"Hello, {name}! You are {age} years old.")
```

-
    - **Run It**: Save as args_test.py, run with python args_test.py Alice 25. Shows how to use sys.argv, like the script's main().

5. **Working with PyPDF2 Library**
    - **What to Know**:
        - PyPDF2 is a Python library for manipulating PDFs; PdfReader reads PDFs, PdfWriter creates/modifies them.
        - Methods like add_page() copy pages, and encrypt() adds password protection.
    - **How It Applies**:
        - The script uses PyPDF2 to read an input PDF, copy its pages, encrypt it, and write the output.
    - **Curriculum Source**:
        - Implied in "PDF Protection Tool (Solution)" Lessons 1-2 (First Function, Finishing our PDF Protector).
        - Note: Students need pip install PyPDF2.
    - **Practical Prep**:
        - Test basic PDF reading with PyPDF2.
    - **Python Code Block**:

```
# Read a PDF and print page count
import PyPDF2

pdf_file = "sample.pdf"  # Replace with a real PDF
try:
    with open(pdf_file, "rb") as f:
        pdf_reader = PyPDF2.PdfReader(f)
        page_count = len(pdf_reader.pages)
```

```
        print(f"{pdf_file} has {page_count} pages")
except FileNotFoundError:
    print(f"{pdf_file} not found")
```

-

    - **Run It**: Install PyPDF2 (pip install PyPDF2), create or download a sample.pdf, save as pdf_test.py, run with python pdf_test.py. Introduces PyPDF2 basics.

6. **Program Entry Point with __name__ == "__main__"**
    - **What to Know**:
        - The if __name__ == "__main__": idiom ensures code runs only when the script is executed directly, not when imported.
    - **How It Applies**:
        - The script uses this to call main(), organizing execution flow.
    - **Curriculum Source**:
        - Not explicitly taught; assumes general Python practice.
    - **Practical Prep**:
        - Test the entry point concept.
    - **Python Code Block**:

```
# Use __name__ == "__main__" for script execution
def say_hello():
    print("Hello from the script!")

if __name__ == "__main__":
    say_hello()
```

    - **Run It**: Save as main_test.py, run with python main_test.py. Shows how to structure a script like the PDF tool.

How PDF Protection Works

- **Concept**:
    - PDF protection adds encryption to a PDF, requiring a password to open it. The script reads an unprotected PDF, copies its pages, applies encryption, and saves the result.
    - This mimics real-world security practices to protect sensitive documents.
- **Script Workflow**:
    - Accept input PDF, output PDF, and password via command-line arguments.
    - Read the input PDF with PyPDF2.PdfReader.
    - Copy pages to a PyPDF2.PdfWriter object.
    - Encrypt with pdf_writer.encrypt(password).
    - Write the encrypted PDF to the output file.
    - Handle errors (e.g., missing file, invalid PDF).
- **Why Command-Line?**:
    - Allows flexible usage without hardcoding file names or passwords.

How to Write the PDF Protection Script

Using these prerequisites, students can build the script step-by-step:

1. **Setup**:
    - Import PyPDF2 and sys.
    - Define cretae_password_protected_pdf with input_pdf, output_pdf, and password parameters.
2. **Read and Process PDF**:
    - Open input_pdf in 'rb' mode, create PdfReader and PdfWriter objects.
    - Loop through pages with len(pdf_reader.pages) and add_page().
3. **Encrypt and Write**:
    - Use encrypt(password) and write to output_pdf in 'wb' mode.
4. **Handle Errors**:
    - Wrap in try/except for FileNotFoundError, PdfReadError, and general Exception.
5. **Main Function**:

    - Check sys.argv length, extract arguments, and call the function.
6. **Entry Point**:
    - Use if __name__ == "__main__": to run main().

Notes for Students

- **Setup**: Install PyPDF2 (pip install PyPDF2) and have a sample PDF ready (e.g., create one with a text editor or download).
- **Engagement**: Run each code block to see file handling, error catching, and PDF manipulation in action. Use a small PDF to test the final script.
- **Command-Line Tip**: Run the final script like python script.py input.pdf output.pdf.

Jump to the PDF Protection Tool Project Description if you are ready now.