**Assignment – 12**

Q1. In what modes should the PdfFileReader() and PdfFileWriter() File objects will be opened?

Sol. When using the **PdfFileReader()** and **PdfFileWriter()** classes from the PyPDF2 library to work with PDF files, the **File** objects should be opened in different modes:

1. **PdfFileReader()**: The **PdfFileReader()** function is used to read and extract information from an existing PDF file. When opening the file with **PdfFileReader()**, you should open it in the "rb" (read binary) mode. This mode ensures that the PDF file is opened in binary mode, allowing proper reading and interpretation of the file's contents.

Example:

file = open('example.pdf', 'rb')

pdf\_reader = PdfFileReader(file)

1. **PdfFileWriter()**: The **PdfFileWriter()** class is used to create a new PDF file or modify an existing one by adding content. When opening the file with **PdfFileWriter()**, you should open it in the "wb" (write binary) mode. This mode ensures that the PDF file is opened in binary mode, allowing proper writing and modification of the file.

Example:

file = open('output.pdf', 'wb')

pdf\_writer = PdfFileWriter()

Q2. From a PdfFileReader object, how do you get a Page object for page 5?

Sol. pdf\_reader = PdfFileReader(file) # Assuming `file` is the opened PDF file

# Get Page object for page 5 (index 4 since page numbering starts from 0)

page\_number = 4 # Zero-based index of the desired page

page = pdf\_reader.getPage(page\_number)

In the above code snippet, **pdf\_reader** is the **PdfFileReader** object that represents the PDF file. The **getPage()** method is called on this object, passing the zero-based index of the desired page (in this case, 4 for page 5).

The resulting **page** variable will hold the Page object for page 5, allowing you to perform various operations on that specific page, such as extracting text, manipulating the content, or obtaining information about the page itself.

Q3. What PdfFileReader variable stores the number of pages in the PDF document?

Sol. In PyPDF2, the PdfFileReader class has a variable named numPages that stores the number of pages in the PDF document. You can access this variable to retrieve the total number of pages in the PDF. Here's an example:

pdf\_reader = PdfFileReader(file) # Assuming `file` is the opened PDF file

num\_pages = pdf\_reader.numPages

In the above code snippet, **pdf\_reader** is the **PdfFileReader** object representing the PDF file. By accessing the **numPages** variable, you can retrieve the total number of pages in the PDF document and assign it to the **num\_pages** variable.

This value can be useful for iterating through all the pages in the PDF, performing operations on specific pages, or obtaining an overview of the document's structure.

Q4. If a PdfFileReader object’s PDF is encrypted with the password swordfish, what must you do before you can obtain Page objects from it?

Sol. Before obtaining Page objects from a PdfFileReader object whose PDF is encrypted with the password "swordfish," you must decrypt the PDF by calling the decrypt() method and passing the password as an argument. Here's an example:

pdf\_reader = PdfFileReader(file) # Assuming `file` is the opened PDF file

if pdf\_reader.isEncrypted:

pdf\_reader.decrypt('swordfish')

# Now we can obtain Page objects from the decrypted PDF

page = pdf\_reader.getPage(page\_number)

pdf\_reader = PdfFileReader(file) # Assuming `file` is the opened PDF file

if pdf\_reader.isEncrypted:

pdf\_reader.decrypt('swordfish')

# Now you can obtain Page objects from the decrypted PDF

page = pdf\_reader.getPage(page\_number)

In the above code snippet, pdf\_reader is the PdfFileReader object representing the PDF file. The isEncrypted property is checked to determine if the PDF is encrypted. If it is encrypted, the decrypt() method is called on the pdf\_reader object, passing the password 'swordfish' as the argument.

Once the PDF is decrypted, you can proceed to obtain Page objects from the PdfFileReader object using methods like getPage(), as shown in the example above. With the PDF decrypted, you can perform various operations on the pages, such as extracting text, modifying content, or accessing page-level information.

Q5. What methods do you use to rotate a page?

Sol.

1. rotateClockwise(): This method rotates the page in a clockwise direction by 90 degrees. Each subsequent call to this method will rotate the page by an additional 90 degrees.
2. rotateCounterClockwise(): This method rotates the page in a counter-clockwise direction by 90 degrees. Each subsequent call to this method will rotate the page by an additional 90 degrees.
3. rotateClockwise90(): This method rotates the page in a clockwise direction by 90 degrees, similar to rotateClockwise(). However, unlike rotateClockwise(), subsequent calls to this method will not rotate the page further.

Q6. What is the difference between a Run object and a Paragraph object?

Sol. In the context of document processing and text manipulation, the difference between a Run object and a Paragraph object depends on the specific library or framework being used. However, in general, the distinction can be described as follows:

**Run Object**: A Run object typically represents a contiguous range of text within a paragraph that has a consistent set of formatting properties. It can be thought of as a "run" or a "span" of text within a paragraph that shares the same formatting characteristics. For example, a Run object may include a sequence of words or even a single character within a paragraph that has the same font style, size, color, or other formatting attributes.

**Paragraph Object**: A Paragraph object represents a block of text that is organized as a single unit or section. It usually contains one or more sentences or lines of text. A Paragraph object provides a way to structure and format text, such as setting alignment, indentation, spacing, and other paragraph-level attributes. It serves as a container for multiple Run objects that collectively form the content of the paragraph.

Q7. How do you obtain a list of Paragraph objects for a Document object that’s stored in a variable named doc?

Sol. To obtain a list of Paragraph objects from a Document object stored in a variable named doc, the specific method or property depends on the library or framework being used. However, in many document processing libraries, you can typically access the Paragraph objects using a method or property that provides access to the content or structure of the document.

Here's a general example using the Python-docx library, which is commonly used for working with Microsoft Word documents:

# Assuming `doc` is the Document object

# Access the paragraphs using the paragraphs property

paragraphs = doc.paragraphs

In the example above, the paragraphs property of the Document object (doc) provides access to a list of Paragraph objects in the document. Each Paragraph object represents a block of text or a paragraph in the document.

Once you have obtained the paragraphs list, you can iterate over it or access specific paragraphs using indexing or other methods provided by the library or framework. The exact methods for accessing and manipulating Paragraph objects may vary depending on the specific document processing library being used.

Q8. What type of object has bold, underline, italic, strike, and outline variables?

Sol.   
The type of object that typically has variables such as bold, underline, italic, strike, and outline is a Run object.

In document processing libraries or frameworks, a Run object represents a contiguous range of text within a paragraph that shares the same formatting properties. It is commonly used to apply formatting attributes to a specific span or "run" of text within a larger paragraph.

The Run object usually has properties or variables that allow you to control various text formatting options, including bold for bold text, underline for underlining, italic for italicizing, strike for strike-through, and outline for an outlined text style. These variables can be set or modified to control the formatting of the text within the Run object.

It's important to note that the specific properties or variables available in a Run object may vary depending on the document processing library or framework being used. The mentioned properties (bold, underline, italic, strike, and outline) are common formatting attributes often found in Run objects.