**Assignment-12**

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

When using the PdfFileReader() function from the PyPDF2 library to read a PDF file and the PdfFileWriter() function to write or modify a PDF file, the File objects should be opened in the following modes:

* PdfFileReader(): The PDF file should be opened in binary read mode ('rb'). This is because PDF files are binary files, and reading them requires binary mode to properly handle the file's internal structure.
* PdfFileWriter(): The PDF file should be opened in binary write mode ('wb'). This is because when using PdfFileWriter(), you typically want to create a new PDF file or modify an existing one by writing data into it.

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

To get a Page object for page 5 from a PdfFileReader object in PyPDF2, you can use the getPage() method and pass the index of the desired page (zero-based index). Since the index starts from 0, you would pass the index 4 to get the Page object for page 5.

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

The PdfFileReader class in PyPDF2 provides a variable called 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 file.

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

To decrypt the PDF, you need to call the decrypt() method of the PdfFileReader object and pass the password as a parameter.

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

To rotate a page in PyPDF2, you can use the rotateClockwise() and rotateCounterClockwise() methods provided by the PageObject class.

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

A Paragraph object represents a paragraph of text in a document. It can contain multiple Run objects and other formatting elements. A paragraph typically represents a block of text that is separated from other paragraphs by a line break or indentation.

A Run object, on the other hand, represents a contiguous run of text within a paragraph. It is a portion of text that shares the same formatting properties, such as font, size, style, color, etc. Within a paragraph, different runs may have different formatting, allowing you to apply specific styles or changes to individual parts of the text.

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

To obtain a list of Paragraph objects from a Document object in python-docx, you can use the paragraphs property of the Document object.

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

The bold, underline, italic, strike, and outline variables are properties of a Run object in the python-docx library.

1. What is the difference between False, True, and None for the bold variable?

In the context of the bold variable in the python-docx library, the values False, True, and None represent different states for the bold formatting property of a Run object.

* False: Setting bold to False means that the run of text should not be displayed in bold. It removes the bold formatting from the text.
* True: Setting bold to True means that the run of text should be displayed in bold. It applies the bold formatting to the text.
* None: If bold is set to None, it indicates that the run of text should inherit the bold formatting from its parent style or from the surrounding text. It doesn't explicitly set the bold property and relies on the default or inherited formatting.

1. How do you create a Document object for a new Word document?

To create a Document object for a new Word document in the python-docx library, you can simply instantiate a new instance of the Document class.

1. How do you add a paragraph with the text ‘Hello, there!’ to a Document object stored in a variable named doc?

To add a paragraph with the text 'Hello, there!' to a Document object stored in a variable named doc, you can use the add\_paragraph() method.

1. What integers represent the levels of headings available in Word documents?

In Word documents, the levels of headings are represented by integers. The common heading levels available in Word documents are:

* Heading 1: Level 1
* Heading 2: Level 2
* Heading 3: Level 3
* Heading 4: Level 4
* Heading 5: Level 5
* Heading 6: Level 6