ASSIGNMENT - 12 (PYTHON)

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

PdfFileReader(): You pass an opened file object in binary read mode ('rb'). This allows PdfFileReader() to read the contents of the PDF file.

PdfFileWriter(): You create an instance of PdfFileWriter() without passing any file object initially. Later, you can add pages and content to it, and then write it to a file using an opened file object in binary write mode ('wb'). This allows PdfFileWriter() to write the PDF contents to a file.

2. 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, you can use the `getPage()` method, which is zero-indexed. Therefore, to get the Page object for page 5 (which is index 4), you would use:

```
page_object = pdf_reader.getPage(4)
```

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

The PdfFileReader variable that stores the number of pages in the PDF document is **`numPages`**.

You can access it as an attribute of the PdfFileReader object. For example:

```
num_pages = pdf_reader.numPages
```

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

If a PdfFileReader object's PDF is encrypted with the password "swordfish," you need to decrypt it before you can obtain Page objects from it.

You can do this by calling the decrypt() method of the PdfFileReader object and passing the password as an argument. Here's how you can do it:

pdf_reader.decrypt('swordfish')

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

rotateClockwise(degrees): Rotates the page clockwise by the specified number of degrees.

page.rotateClockwise(90) # Rotate the page 90 degrees clockwise

rotateCounterClockwise(degrees): Rotates the page counter-clockwise by the specified number of degrees.

page.rotateCounterClockwise(90) # Rotate the page 90 degrees counter-clockwise

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

Run Object:

A Run object represents a contiguous run of text within a paragraph that has the same set of properties (such as font size, font family, bold, italic, etc.).

Runs are the smallest units of text manipulation in python-docx. They allow you to apply formatting to specific parts of a paragraph without affecting the entire paragraph.

Runs can contain text, as well as formatting properties like bold, italic, underline, font size, font color, etc.

You can access and manipulate individual runs within a paragraph.

Paragraph Object:

A Paragraph object represents a block of text within a document, separated by line breaks or paragraph breaks.

Paragraphs are containers for one or more Run objects. They represent a complete unit of text within a document.

Paragraphs can contain various formatting properties, such as alignment, indentation, spacing, etc., which apply to the entire block of text.

You can access and manipulate paragraphs within a document, including adding or removing text, applying formatting, and adjusting paragraph properties.

7. 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 for a Document object stored in a variable named doc using the python-docx library, you can access the paragraphs attribute of the Document object. Here's how you do it:

paragraphs = doc.paragraphs

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

The object type that typically has attributes like **`bold`**, **`underline`**, **`italic`**, **`strike`**, and **`outline`** in the context of document manipulation libraries like **`python-docx`** is a Run object.

In **`python-docx`**, a Run object represents a contiguous run of text within a paragraph that has the same set of properties (such as formatting). These properties include

attributes like `bold`, `underline`, `italic`, `strike`, `outline`, etc., which determine the appearance of the text represented by the Run object

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

False:

Setting bold to False means the text will not be displayed in bold.

Example: run.bold = False

True:

Setting bold to True means the text will be displayed in bold.

Example: run.bold = True

None:

Setting bold to None means that the text will inherit the boldness from the surrounding context. If the surrounding context has no specific boldness applied, the text will not be displayed in bold.

Example: run.bold = None

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

To create a `Document` object for a new Word document using the `python-docx` library, you can use the `Document()` constructor. Here's how you do it:

```
from docx import Document

doc = Document()
```

11. 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 of the Document object. Here's how you do it:

```
from docx import Document

doc = Document()

doc.add_paragraph('Hello, there!')
```

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

In Word documents, headings are typically organized into levels, with each level indicating a different level of hierarchy in the document structure. In python-docx, these levels are represented by integers. The integers representing the levels of headings available in Word documents are:

Level 1: Integer value 0

Level 2: Integer value 1

Level 3: Integer value 2

Level 4: Integer value 3

Level 5: Integer value 4

Level 6: Integer value 5

Level 7: Integer value 6

Level 8: Integer value 7

Level 9: Integer value 8

These integers are used to specify the level of the heading when adding headings to a Word document using python-docx. For example, to add a heading with level 1, you would use doc.add_heading('Heading Text', level=0).