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

In Python, when using the PdfFileReader and PdfFileWriter classes from the PyPDF2 library, the PDF file should be opened in binary mode ('rb' for reading, 'wb' for writing).

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

From a PdfFileReader object, you can get a Page object for page 5 by using the getPage method:

from PyPDF2 import PdfFileReader

with open('input.pdf', 'rb') as file:

pdf\_reader = PdfFileReader(file)

page\_5 = pdf\_reader.getPage(4) # Page numbers start from 0, so page 5 is at index 4

After obtaining the Page object, you can perform operations such as extracting text, rotating the page, or adding annotations to it.

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

In Python, the PdfFileReader class from the PyPDF2 library has a numPages attribute that stores the number of pages in the PDF document. You can access this attribute after creating a PdfFileReader object

**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 must provide the correct password 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:

from PyPDF2 import PdfFileReader

with open('input.pdf', 'rb') as file:

pdf\_reader = PdfFileReader(file)

if pdf\_reader.isEncrypted:

pdf\_reader.decrypt("swordfish")

page\_5 = pdf\_reader.getPage(4)

If the password is incorrect, the decrypt method will raise a NotImplementedError exception.

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

To rotate a page in Python using the PyPDF2 library, you can use the rotateClockwise method of the Page object. The method takes an angle in degrees as an argument and rotates the page in a clockwise direction. For example, to rotate a Page object 90 degrees clockwise:

from PyPDF2 import PdfFileReader

with open('input.pdf', 'rb') as file:

pdf\_reader = PdfFileReader(file)

page\_5 = pdf\_reader.getPage(4)

page\_5.rotateClockwise(90)

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

The Run and Paragraph objects are part of the python-docx library, which allows you to create, read, and modify Microsoft Word documents.

A Paragraph object represents a paragraph in a Word document. It has properties such as font, style, and alignment, and you can add runs of text to it. A Run object represents a contiguous sequence of characters with the same font, style, and formatting within a Paragraph.

For example, consider a paragraph with the text "This is bold and italic." To format the word "bold" in bold and the word "italic" in italic, you would create two Run objects within the same Paragraph object

**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, you can use the paragraphs property of the Document object:

from docx import Document

doc = Document('input.docx')

paragraphs = doc.paragraphs

The paragraphs property returns a list of Paragraph objects, each representing a paragraph in the document. You can then iterate over the list to access the properties and methods of each Paragraph object

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

In the python-docx library, the Run object has the bold, underline, italic, strike, and outline variables. A Run object represents a contiguous sequence of characters with the same font, style, and formatting within a paragraph in a Microsoft Word document. These variables allow you to set the font properties of a Run object, such as making the text bold, italic, underlined, struck through, or outlined.

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

In the context of the Run object in the python-docx library, the bold variable is a boolean value that determines whether the text of the Run object should be bold or not.

The difference between False, True, and None values for the bold variable is as follows:

* False: If bold is set to False, the text of the Run object will not be bold.
* True: If bold is set to True, the text of the Run object will be bold.
* None: If bold is set to None, the text of the Run object will not have a bold setting and will inherit the bold setting from the parent Paragraph object or the default style.

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

To create a new Microsoft Word document in python, you can use the Document class from the python-docx library. Here's an example of how to create a new Document object:

from docx import Document

doc = Document()

doc.save('output.docx')

This creates a new Document object and saves it as a Microsoft Word document with the name 'output.docx'. You can then add paragraphs, tables, images, or other elements to the document by calling methods on the Document object.

**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:

from docx import Document

doc = Document()

paragraph = doc.add\_paragraph('Hello, there!')

doc.save('output.docx')

This creates a new paragraph with the text 'Hello, there!' and adds it to the Document object stored in the doc variable. The resulting Microsoft Word document is then saved to a file with the name 'output.docx'.

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

In the python-docx library, headings in a Microsoft Word document are represented by the Paragraph object and can be assigned a level from 1 to 9, inclusive.

The level of a heading is determined by the style applied to the Paragraph object. The built-in heading styles in Microsoft Word are Heading 1 to Heading 9, and they can be applied to a Paragraph object in python-docx as follows:

from docx import Document

doc = Document()

# Add a heading of level 1 (Heading 1)

heading1 = doc.add\_paragraph('Heading 1', style='Heading 1')

# Add a heading of level 2 (Heading 2)

heading2 = doc.add\_paragraph('Heading 2', style='Heading 2')

# Add a heading of level 3 (Heading 3)

heading3 = doc.add\_paragraph('Heading 3', style='Heading 3')

# Add a heading of level 9 (Heading 9)

heading9 = doc.add\_paragraph('Heading 9', style='Heading 9')

doc.save('output.docx')

In this example, a Paragraph object is created for each heading level and assigned the corresponding heading style ('Heading 1', 'Heading 2', 'Heading 3', etc.). The resulting Microsoft Word document will have the headings formatted according to the styles you assigned.