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

ANS :- The `PdfFileReader()` function should open the PDF file in "rb" mode, which stands for "read binary." This mode is used to read the contents of a PDF file.

The `PdfFileWriter()` function should open the PDF file in "wb" mode, which stands for "write binary." This mode is used to write data and create a new PDF file or modify an existing one.

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

ANS :- To get a `Page` object for page 5 from a `PdfFileReader` object, you can use the `getPage()` method and pass the index of the desired page, which starts from 0. In this case, WE would use `getPage(4)` to get the `Page` object for page 5 since the index is zero-based.

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

ANS :- The `PdfFileReader` variable that stores the number of pages in the PDF document is `numPages`. You can access it using the dot notation with the `PdfFileReader` object. For example, if `pdf\_reader` is your `PdfFileReader` object, you can get the number of pages with `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?

ANS :- If a `PdfFileReader` object's PDF is encrypted with the password "swordfish", we need to decrypt the PDF using the `decrypt()` method before obtaining `Page` objects from it. We need to pass the correct password as an argument to the `decrypt()` method. Here's an example:

```python

pdf\_reader = PdfFileReader('encrypted.pdf')

pdf\_reader.decrypt('swordfish')

# Now you can access Page objects from the decrypted PDF

page = pdf\_reader.getPage(0)

```

Once the PDF is successfully decrypted, you can obtain `Page` objects using the `getPage()` method or iterate over the pages using a loop.

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

ANS :- To rotate a page in PyPDF2, you can use the `rotateClockwise()` and `rotateCounterClockwise()` methods of a `Page` object. Here's an example:

```python

from PyPDF2 import PdfFileReader, PdfFileWriter

# Open the PDF file

pdf\_reader = PdfFileReader('input.pdf')

# Get the first page

page = pdf\_reader.getPage(0)

# Rotate the page clockwise by 90 degrees

page.rotateClockwise(90)

# Create a new PDF file

pdf\_writer = PdfFileWriter()

pdf\_writer.addPage(page)

# Save the rotated page to a new file

with open('output.pdf', 'wb') as file:

pdf\_writer.write(file)

```

In this example, we open the input PDF file, get the first page using `getPage()`, rotate the page clockwise by 90 degrees using `rotateClockwise()`, and then create a new PDF file to save the rotated page using `PdfFileWriter`. Finally, we write the modified PDF to the output file.

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

ANS :- A Paragraph is like a container for text. It represents a block of text, like a paragraph in a document. It can have multiple lines of text and can have its own formatting, like alignment and style.

A Run, on the other hand, is like a piece of text within a Paragraph. It's a smaller unit of text that can have its own formatting, like font size, color, and style. Runs allow you to apply different styles or formatting to specific parts of the text within a paragraph.

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

ANS :- To obtain a list of Paragraph objects from a Document object stored in the variable named `doc`, we can use the `doc.paragraphs` property. Here's an example:

```python

paragraphs = doc.paragraphs’’’

This will give you a list of Paragraph objects, where each element in the list represents a paragraph in the document. You can then iterate over this list to access and work with individual paragraphs as needed.

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

ANS :- The object that has `bold`, `underline`, `italic`, `strike`, and `outline` variables is a Run object. In the context of document processing libraries like `python-docx`, a Run object represents a contiguous run of text within a paragraph that has consistent formatting properties. The Run object allows you to access and modify the formatting attributes of the text it represents, such as bold, underline, italic, strike-through, and outline styles.

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

ANS :- In the context of document processing libraries like `python-docx`, the `bold` variable in a Run object can have three possible values: `False`, `True`, or `None`. Here's what each value represents:

1. `False`: If `bold` is set to `False`, it means the text in the Run object is not formatted as bold. It indicates that the text should have a regular or normal font weight.

2. `True`: If `bold` is set to `True`, it means the text in the Run object is formatted as bold. It indicates that the text should have a bold font weight.

3. `None`: If `bold` is set to `None`, it means the text in the Run object inherits the bold formatting from its parent style or formatting settings. In this case, the actual boldness of the text may depend on the underlying style applied to the paragraph or the document's default settings.

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

ANS :- To create a `Document` object for a new Word document using the `python-docx` library, we can simply call the `Document()` constructor. Here's an example:

```python

from docx import Document

# Create a new document object

doc = Document()

# Now you can work with the 'doc' object to add content, styles, etc.

# For example, you can add paragraphs, tables, images, etc.

# Save the document

doc.save('new\_document.docx')

```

In the above example, `Document()` creates a new, empty `Document` object, which represents a new Word document. You can then use this object to add paragraphs, tables, images, and other content to the document. Finally, the `save()` method is used to save the document to a specified file name, in this case, 'new\_document.docx'.

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

ANS :- 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. Here's an example:

```python

from docx import Document

# Create a new document object

doc = Document()

# Add a paragraph with the text 'Hello, there!'

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

# Save the document

doc.save('new\_document.docx')

```

In the above example, `add\_paragraph()` is called on the `doc` object to add a new paragraph with the specified text. The `save()` method is then used to save the document to a file named 'new\_document.docx'.

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

ANS :- In Word documents, the levels of headings are represented by integers ranging from 1 to 9. Each integer corresponds to a specific heading level, where 1 represents the highest level (main heading) and 9 represents the lowest level (sub-sub-sub-sub-sub-sub-sub-subheading). Here's a summary of the heading levels and their corresponding integers:

- Heading 1: Level 1

- Heading 2: Level 2

- Heading 3: Level 3

- Heading 4: Level 4

- Heading 5: Level 5

- Heading 6: Level 6

- Heading 7: Level 7

- Heading 8: Level 8

- Heading 9: Level 9

These heading levels are used to create a structured hierarchy within the document, allowing for easy navigation and formatting.