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

In Python's `PyPDF2` library, the `PdfFileReader()` and `PdfFileWriter()` objects should be opened in different modes depending on the operation you want to perform:

1. `PdfFileReader()` object: This is used to read or extract information from an existing PDF file. It should be opened in "**rb**" mode, which stands for "read binary." Opening the PDF file in binary mode is necessary because **PDF files contain binary data**.

```

from PyPDF2 import PdfFileReader

pdf\_reader = PdfFileReader(open('example.pdf', 'rb'))

```

2. `PdfFileWriter()` object: This is used to create a new PDF file or modify an existing one by adding pages or other content. It should be opened in "**wb**" mode, which stands for "write binary." Opening the PDF file in binary mode ensures that the output is written correctly.

```

from PyPDF2 import PdfFileWriter

pdf\_writer = PdfFileWriter()

output\_pdf = open('output.pdf', 'wb')

pdf\_writer.write(output\_pdf)

output\_pdf.close()

```

**In both cases, the file objects are opened using the `open()` function with the appropriate mode. The "rb" mode is used for reading an existing PDF file, while the "wb" mode is used for writing or modifying a PDF file.**

**Q. 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 and pass the page index as an argument. In Python's `PyPDF2` library, the page index starts from 0 for the first page.

Here's an example:

```

from PyPDF2 import PdfFileReader

pdf\_reader = PdfFileReader(open('example.pdf', 'rb'))

# Get Page object for page 5 (index 4)

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

```

In this example, `**pdf\_reader.getPage(4)` retrieves the `Page` object for page 5 (index 4) from the `PdfFileReader` object**. **You can then perform operations** **on this `Page` object, such as extracting text or modifying the page content.**

**Q. 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`. It is an attribute of the `PdfFileReader` object that represents the total number of pages in the PDF file.

Here's an example:

```

from PyPDF2 import PdfFileReader

pdf\_reader = PdfFileReader(open('example.pdf', 'rb'))

# Get the number of pages in the PDF document

**num\_pages = pdf\_reader.numPages**

print("Number of pages:", num\_pages)

```

In this example, `pdf\_reader.numPages` retrieves the total number of pages in the PDF document and assigns it to the `num\_pages` variable. You can then use `num\_pages` to perform operations or iterate over the pages of the PDF document.

**Q. 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 using the `decrypt()` method before obtaining `Page` objects from it.**

Here's an example:

```

from PyPDF2 import PdfFileReader

pdf\_reader = PdfFileReader(open('encrypted.pdf', 'rb'))

# Decrypt the PDF with the password

**pdf\_reader.decrypt('swordfish')**

# Now you can access the Page objects

page = pdf\_reader.getPage(0)

```

In this example, **`pdf\_reader.decrypt('swordfish')` decrypts the PDF document using the provided password "swordfish." Once the PDF is decrypted, you can access the `Page` objects using the `getPage()` method or perform other operations on the PDF document**.

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

To rotate a page in PyPDF2, you can use the `**rotateClockwise()`** or `**rotateCounterClockwise()`** methods of the `Page` object. These methods allow you to rotate the page by **90 degrees in the clockwise or counter-clockwise direction, respectively**.

Here's an example:

```

from PyPDF2 import PdfFileReader, PdfFileWriter

pdf\_reader = PdfFileReader(open('input.pdf', 'rb'))

pdf\_writer = PdfFileWriter()

page = pdf\_reader.getPage(0)

# Rotate the page 90 degrees clockwise

**page.rotateClockwise(90)**

# **Add the rotated page to the output PDF writer**

pdf\_writer.addPage(page)

# **Save the modified PDF to a new file**

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

pdf\_writer.write(output\_file)

```

In this example, `rotateClockwise(90)` is used to rotate the page 90 degrees clockwise. You can replace it with `rotateCounterClockwise(90)` to rotate the page in the counter-clockwise direction. **After rotating the page, it is added to a `PdfFileWriter` object, which can then be saved to a new PDF file using the `write()` method**.

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

In the context of Python's `python-docx` library, a `Run` object and a `Paragraph` object represent different elements of a Word document:

1. \*\*Paragraph object\*\*: A `Paragraph` object represents a paragraph in a Word document. It consists of one or more runs of text and can contain formatting properties such as font style, size, alignment, indentation, etc. A paragraph is typically a block of text with line breaks.

2. \*\*Run object\*\*: A `Run` object represents a contiguous run of text within a paragraph. It is a segment of text that has the same formatting properties within a paragraph. For example, if a paragraph has a sentence with a different font style or size, that sentence would be represented as a separate run within the paragraph. Runs allow you to apply formatting changes to specific portions of text within a paragraph.

To summarize, a `Paragraph` object represents a whole paragraph, while a `Run` object represents a segment of text within that paragraph. Paragraphs can contain one or more runs, and runs allow for more granular control over formatting changes within a paragraph.

Q. **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. Here's an example:

```

from docx import Document

# Assuming you have a Document object stored in the variable 'doc'

doc = Document("path/to/document.docx")

**# Get a list of Paragraph objects**

**paragraphs = doc.paragraphs**

**# Iterate over the paragraphs**

**for paragraph in paragraphs:**

**# Perform operations on each paragraph**

**print(paragraph.text)**

```

In this example, the `paragraphs` property of the `Document` object returns a list of all the paragraphs in the document. You can then iterate over the list and access each `Paragraph` object to perform operations or access properties such as the text content using the `text` attribute (`paragraph.text` in the example).

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

The `**Run` object** in the **`python-docx` library** has the variables `bold`, `underline`, `italic`, `strike`, and `outline` that represent different formatting options for text.

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

In the `python-docx` library, the `bold` variable in a `Run` object can take three different values:

- `False`: It indicates that the text is not bold.

- `True`: It indicates that the text is bold.

- `None`: It indicates that the bold setting for the text is not explicitly specified. In this case, the text may inherit the bold setting from a style applied to it or from the default formatting of the document.

In summary, `False` means not bold, `True` means bold, and `None` means the bold setting is not explicitly specified.

Q. **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 following code:

```

**from docx import Document**

**doc = Document()**

```

This **code initializes a new `Document` object, which represents an empty Word document.** You can then manipulate this object to add content, formatting, and other elements to the document.

Q. **How do you add a paragraph with the text &#39;Hello, there!&#39; 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 following code:

```

from docx import Document

doc = Document()

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

```

**This code creates a new `Document` object** and **adds a paragraph with the specified text using the `add\_paragraph` method.** The **paragraph is appended to the document**.

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

In Word documents, the **integers 0 to 9 represent the levels of headings** available. The **higher the integer, the lower the heading level**. Here's a brief overview of the common heading levels:

- Heading 1: Level 1 (integer: 0)

- Heading 2: Level 2 (integer: 1)

- Heading 3: Level 3 (integer: 2)

- Heading 4: Level 4 (integer: 3)

- Heading 5: Level 5 (integer: 4)

- Heading 6: Level 6 (integer: 5)

- Heading 7: Level 7 (integer: 6)

- Heading 8: Level 8 (integer: 7)

- Heading 9: Level 9 (integer: 8)

The exact number of available heading levels may depend on the version of Word or the specific document template being used.