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

The PdfFileReader() function from the PyPDF2 library and the PdfFileWriter() function both operate on PDF files. These functions expect the PDF files to be opened in different modes:

PdfFileReader(): The PDF file opened with PdfFileReader() should be opened in binary read mode ('rb'). This is because PDF files are binary files, and reading them requires binary mode to correctly handle the binary data within the file.

Eg:

pdf\_file = open('example.pdf', 'rb')

pdf\_reader = PdfFileReader(pdf\_file)

PdfFileWriter(): The PDF file opened with PdfFileWriter() should be opened in binary write mode ('wb'). This is because PdfFileWriter() is used to create or modify PDF files, and writing binary data to the file requires binary write mode.

Eg:

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

pdf\_writer = PdfFileWriter()

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

from PyPDF2 import PdfFileReader

pdf\_file = open('example.pdf', 'rb')

pdf\_reader = PdfFileReader(pdf\_file)

page\_5 = pdf\_reader.getPage(4) # Get Page object for page 5 (index 4)

# You can now perform operations on the page, such as extracting text or modifying it

# For example, to get the text content of the page:

page\_5\_text = page\_5.extract\_text()

print(page\_5\_text)

pdf\_file.close()

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

The PdfFileReader class in the PyPDF2 library has a variable called numPages that stores the number of pages in the PDF document.

Eg:

from PyPDF2 import PdfFileReader

pdf\_file = open('example.pdf', 'rb')

pdf\_reader = PdfFileReader(pdf\_file)

num\_pages = pdf\_reader.numPages

print(f"Number of pages: {num\_pages}")

pdf\_file.close()

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 PDF file encrypted with the password "swordfish" is opened with a PdfFileReader object, you need to decrypt the PDF file using the decrypt() method before you can obtain Page objects from it.

Eg:

from PyPDF2 import PdfFileReader

pdf\_file = open('encrypted.pdf', 'rb')

pdf\_reader = PdfFileReader(pdf\_file)

if pdf\_reader.isEncrypted:

pdf\_reader.decrypt('swordfish')

# Now you can access the Page objects from the decrypted PDF

num\_pages = pdf\_reader.numPages

for page\_number in range(num\_pages):

page = pdf\_reader.getPage(page\_number)

# Do something with the page

pdf\_file.close()

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

To rotate a page in PyPDF2, you can use the rotateClockwise() and rotateCounterClockwise() methods of a Page object. These methods allow you to rotate the page clockwise or counterclockwise by a specified angle.

Eg:

from PyPDF2 import PdfFileReader, PdfFileWriter

# Open the PDF file

pdf\_file = open('input.pdf', 'rb')

pdf\_reader = PdfFileReader(pdf\_file)

# Create a new PDF writer

pdf\_writer = PdfFileWriter()

# Rotate the first page clockwise by 90 degrees

page = pdf\_reader.getPage(0)

page.rotateClockwise(90)

# Add the rotated page to the PDF writer

pdf\_writer.addPage(page)

# Save the rotated PDF to a new file

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

pdf\_writer.write(output\_pdf)

# Close the files

pdf\_file.close()

output\_pdf.close()

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

In the context of python-docx library, a Run object represents a contiguous run of text within a paragraph that has a consistent set of character formatting properties. It is essentially a span of text with specific formatting such as font style, size, color, etc. A Run object can contain plain text or a combination of formatted text.

On the other hand, a Paragraph object represents a single paragraph of text within a document. It consists of one or more Run objects. A Paragraph object can contain multiple Runs, each with its own formatting properties. It represents a complete block of text with its associated formatting, such as alignment, indentation, spacing, etc.

In summary, a Run object represents a portion of text with specific formatting properties, while a Paragraph object represents a complete paragraph with its formatting and contains one or more Run objects.

Eg:

from docx import Document

# Create a new document

doc = Document()

# Add a paragraph with formatted text

paragraph = doc.add\_paragraph()

run1 = paragraph.add\_run("Hello, ")

run1.bold = True

run1.italic = True

run1.font.size = 14

run2 = paragraph.add\_run("world!")

run2.font.color.rgb = docx.shared.RGBColor(0x42, 0x24, 0xE9)

# Add another paragraph

paragraph2 = doc.add\_paragraph("This is another paragraph.")

# Save the document

doc.save("example.docx")

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

from docx import Document

# Load the document

doc = Document("example.docx")

# Get the list of paragraphs

paragraphs = doc.paragraphs

# Iterate over the paragraphs

for paragraph in paragraphs:

print(paragraph.text)

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

The Run object in python-docx has the variables bold, underline, italic, strike, and outline that represent formatting attributes for text.

Eg:

from docx import Document

# Create a new document

doc = Document()

# Add a paragraph with formatted text

paragraph = doc.add\_paragraph()

run = paragraph.add\_run("This is some text")

run.bold = True

run.underline = True

run.italic = True

run.strike = True

run.outline = True

# Save the document

doc.save("formatted\_text.docx")

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

In the context of the bold variable in python-docx:

* False means that the text is not formatted as bold.
* True means that the text is formatted as bold.
* None means that the bold formatting is inherited from the style applied to the text. If no explicit bold formatting is applied to the text or the style, then None indicates that the default formatting should be used.

Eg:

from docx import Document

# Create a new document

doc = Document()

# Add a paragraph with different bold settings

paragraph = doc.add\_paragraph()

run1 = paragraph.add\_run("This is not bold")

run1.bold = False

run2 = paragraph.add\_run("This is bold")

run2.bold = True

run3 = paragraph.add\_run("This uses the default bold setting")

run3.bold = None

# Save the document

doc.save("bold\_text.docx")

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

To create a Document object for a new word document using a python doc library and by using Document() constructor.

Eg:

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?

from docx import Document

doc = Document()

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

doc.save(“document.docx”)

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

In Word documents, the following integers represent the levels of headings:

* Level 1: 0
* Level 2: 1
* Level 3: 2
* Level 4: 3
* Level 5: 4
* Level 6: 5
* Level 7: 6
* Level 8: 7
* Level 9: 8

These integers correspond to the heading levels available in Word documents, where level 1 is the highest and level 9 is the lowest. You can use these integers when working with the add\_heading() method of the Document object in the python-docx library to specify the desired heading level.