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

Ans:-

For PdfFileReader():

import PyPDF2

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

pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

PdfFileWriter():

import PyPDF2

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

pdf\_writer = PyPDF2.PdfFileWriter()

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

Ans:-

To get a Page object for a specific page in a PDF file using the PdfFileReader object from PyPDF2, you can use the getPage() method and pass the page number (starting from 0) as an argument.

import PyPDF2

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

pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

page\_number = 4 # Page numbers start at 0, so page 5 is number 4

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

# Do something with the page object

pdf\_file.close()

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

Ans:-

In PyPDF2, the PdfFileReader class has a property called numPages, which stores the number of pages in the PDF document. You can access this property on a PdfFileReader object to get the total number of pages in the document.

import PyPDF2

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

pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

num\_pages = pdf\_reader.numPages

print('The PDF document contains', num\_pages, '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?

Ans:- Before obtaining Page objects from an encrypted PDF using a PdfFileReader object, you must first decrypt the PDF by calling the decrypt() method on the PdfFileReader object and passing in the password as a string argument.

import PyPDF2

# Open the encrypted PDF file in read-binary mode

with open('encrypted.pdf', 'rb') as encrypted\_file:

# Create a PdfFileReader object

pdf\_reader = PyPDF2.PdfFileReader(encrypted\_file)

# Decrypt the PDF with the password 'swordfish'

pdf\_reader.decrypt('swordfish')

# Check if the PDF is decrypted

if pdf\_reader.isEncrypted:

print('Failed to decrypt PDF file')

else:

# Access Page objects from the decrypted PDF

page = pdf\_reader.getPage(0)

# Do something with the Page object

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

Ans:-

To rotate a page in a PDF file using the PyPDF2 library in Python, you can use the rotateClockwise() and rotateCounterClockwise() methods of the PageObject class. These methods rotate the page 90 degrees clockwise or counterclockwise, respectively.

import PyPDF2

# Open a PDF file in read-binary mode

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

# Create a PdfFileReader object

pdf\_reader = PyPDF2.PdfFileReader(input\_file)

# Get the first page of the PDF file

page = pdf\_reader.getPage(0)

# Rotate the page 90 degrees clockwise

page.rotateClockwise(90)

# Create a PdfFileWriter object

pdf\_writer = PyPDF2.PdfFileWriter()

# Add the rotated page to the writer object

pdf\_writer.addPage(page)

# Open an output PDF file in write-binary mode

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

# Write the rotated page to the output PDF file

pdf\_writer.write(output\_file)

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

Ans:-

a Paragraph object represents a single paragraph of text, while a Run object represents a contiguous run of characters within a paragraph that all share the same set of character-level formatting, such as font style, font size, color, and so on.

A Paragraph object in Word can contain multiple Run objects, but a Run object cannot contain other Run objects or Paragraph objects. Each Paragraph object starts with a new line and can be separated from the previous paragraph by a blank line or other paragraph formatting.

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 for a Document object in python-docx that is stored in a variable named doc, you can use the paragraphs attribute of the Document object.

import docx

# Open an existing Word document

doc = docx.Document('example.docx')

# Get a list of Paragraph objects in the document

paragraphs = doc.paragraphs

# Print the text of each paragraph in the document

for paragraph in paragraphs:

print(paragraph.text)

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

Ans:-

In python-docx, the Run object has the bold, underline, italic, strike, and outline variables that can be used to modify the formatting of text in a Word document.

A Run object represents a contiguous run of characters within a paragraph that all share the same set of character-level formatting, such as font style, font size, color, and so on. The bold, underline, italic, strike, and outline variables are Boolean attributes of the Run object that determine whether the text in the run is bold, underlined, italicized, struck through, or outlined, respectively.

import docx

# Open an existing Word document

doc = docx.Document('example.docx')

# Get the first paragraph in the document

paragraph = doc.paragraphs[0]

# Get the first run in the paragraph

run = paragraph.runs[0]

# Modify the formatting of the run

run.bold = True

run.underline = True

# Save the document

doc.save('example.docx')

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

* When bold is set to True, the text in the Run object will be displayed in bold font.
* When bold is set to False, the text in the Run object will be displayed in regular font (i.e., not bold).
* When bold is set to None, the text in the Run object will inherit the bold formatting from its parent style, which can be either the underlying paragraph style or an explicit character style applied to the Run object.

import docx

# Open an existing Word document

doc = docx.Document('example.docx')

# Get the first paragraph in the document

paragraph = doc.paragraphs[0]

# Get the first run in the paragraph

run = paragraph.runs[0]

# Set bold to True

run.bold = True

# Set bold to False

run.bold = False

# Set bold to None

run.bold = None

# Save the document

doc.save('example.docx')

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

Ans:-

import docx

# Create a new Document object

doc = docx.Document()

# Add a paragraph to the document

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

# Save the document

doc.save('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 of the Document object. Here's an example code snippet that demonstrates how to do this:

import docx

# Open an existing Word document

doc = docx.Document('existing\_document.docx')

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

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

# Save the document

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

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

Ans:-

‘1’: Heading 1

‘2’: Heading 2

‘3’: Heading 3

‘4’: Heading 4

These integers correspond to the different heading styles in Word, which are used to apply consistent formatting to headings of different levels. In python-docx, you can apply these heading styles to a paragraph using the style attribute of the Paragraph object. For example, to apply Heading 1 to a paragraph, you would set the style attribute to 'Heading 1':

import docx

# Create a new Document object

doc = docx.Document()

# Add a Heading 1 paragraph to the document

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

# Add a Heading 2 paragraph to the document

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

# Save the document

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