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

**Ans:-** When using PyPDF2 library in python, the **‘PdfFileReader()’** and **‘PdfFileWriter()’** objects are used to read and write PDF files respectively. This object requires a file object to be passed as an argument to their constructor.

1. **PdfFileReader() :-** The file object should be opened in binary mode with the ‘rb’ mode , which stands for “read binary”. This is because PDF files are binary files, and opening them in text mode can result in data corruption or loss.

pdf\_file = open('file\_path.pdf', 'rb')

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

1. **PdfFileWriter() :-** The file object should be opened in binary mode with the ‘wb’ mode, which stands for ‘write binary’. This is because the ‘PdfFileWriter()’ object writes binary data to the pdf file. So the file object should be opened like this;-

pdf\_file = open('file\_path.pdf', 'wb')

pdf\_writer = PyPDF2.PdfFileWriter()

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

**Ans;-** We can use the “getpage()” method of the “PdfFileReader()” object to get a “page” object for a specific page. To get a “page” object for page 5, we can use the following code for this:-

from PyPDF2 import PdfFileReader

# Open the PDF file

with open('example.pdf', 'rb') as f:

# Create a PdfFileReader object

pdf\_reader = PdfFileReader(f)

# Get the Page object for page 5

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

The “getpage()” method uses a 0-based index, so the first page is at index 0, the second page is at index 1, and so on.

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

**Ans:-** In “PyPDF2”, The “PdfFileReader()” object has a variable named ‘numpages’ that stores the number of pages in the PDF document. You can access this variable as follows:-

from PyPDF2 import PdfFileReader

# Open the PDF file

with open('example.pdf', 'rb') as f:

# Create a PdfFileReader object

pdf\_reader = PdfFileReader(f)

# Get the number of pages in the PDF document

num\_pages = pdf\_reader.numPages

In this example, ‘num\_pages’ will contain the number of pages in the PDF document.

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:-** To obtain the page objects from the PDF file encrypted with a password using python’s ‘PdfFileReader’ module, you must first decrypt the PDF file by providing the correct password. Here’s an example of how to do it:-

import PyPDF2

# Open the encrypted PDF file in read-binary mode

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

# Create a PdfFileReader object

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

# Check if the PDF file is encrypted

if pdf\_reader.isEncrypted:

# Decrypt the PDF file using the password 'swordfish'

pdf\_reader.decrypt('swordfish')

# Get the number of pages in the PDF file

num\_pages = pdf\_reader.getNumPages()

# Access individual Page objects from the PDF file

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

# Get the Page object for the current page

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

# Do something with the Page object

# For example, you could extract text from the page using page.extractText()

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

**Ans:-** To rotate a page in a PDF file using python, you can use the PyPF2 library’s “rotateclockwise()” and “rotatecounterclockwise()” methods of “pageobject” class. Here’s an example of how to do it:-

import PyPDF2

# Open the PDF file in read-binary mode

with open('input\_file.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 clockwise by 90 degrees

page.rotateClockwise(90)

# Save the rotated page to a new PDF file

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

# Create a PdfFileWriter object

pdf\_writer = PyPDF2.PdfFileWriter()

# Add the rotated page to the PdfFileWriter object

pdf\_writer.addPage(page)

# Write the PdfFileWriter object to the output file

pdf\_writer.write(output\_file)

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

**Ans:-** In python’s “python-docx” library, a “paragraph” object represents a single paragraph in a word document, while a “run” object represents a contiguous run of text within a paragraph that has the same character formatting. Here’s some detailed explanation:-

1. **Paragraph object :-** A “paragraph” objects represents a single paragraph in a word document. It contains one or more “run” objects that represents the runs of text within the paragraph. A “paragraph” object can have its own paragraph formatting, such as indentation, line spacing and alignment. You can create a new paragraph object using the “add\_paragraph” method of a “document” object or by accessing an existing “paragraph” object using the “paragraph” attribute of a “document” object.
2. **Run object :-** A “run” object represennts a contiguous run of text within a “paragraph” object that has the same character formatting. A “run” object can have its own character formatting, such as fon size, font style and font color. You can create a new “run” object using the “add\_run()” method of a “paragraph” object or by accessing an existing “run” object using the “run” attribute of a “paragraph” object.

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

**Ans:-** You can obtain a list of ‘paragraph’ objects from a “document” object by calling the “paragraph” attribute of the ‘document” object. Here’s an example:-

import docx

# Load the document

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

# Get a list of all the paragraphs in the document

paragraphs = doc.paragraphs

# Print the text of each paragraph

for paragraph in paragraphs:

print(paragraph.text)

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

**Ans:-** In python, the “python-docx” module allow you to manipulate microsoft word files programmatically, including their formatting. The “docx.text.run” module, provides a “run” object that has attribute to control the formatting of the text in a word document. The “run” object has the following attributes that can be used to control the text formatting.

1. **bold :-** A boolean value that specifies whether the text should be bold or not.
2. **Underline :-** A member of the “docx.enum.text.WD\_UNDERLINE” enumeration that specifies the type of the underline to apply to the text.
3. **Italic :-** A boolean value that specifies whether the text should be italic or not.
4. **Strike :-** A boolean value that specifies whether the text should have the strikethrough” or not.
5. **Outline :-** A boolean value that specifies whether the text should be outlined or not.

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

**Ans:-** In python, False, True, and None are the reserved keywords that represents the specific values.

1. False represents a boolean false value. It is used to represent the absence of a condition or to indicate that the condition is not true.
2. True represents a boolean true value. It is used to represent the presence of a condition or to indicate that a condition is true.
3. None is built-in constant that represents the absence of a value, It is often used to represent the missing or undefined data.

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

**Ans:-** You can create a new word documen using the ‘docx’ module in python. Here’s an example of how to create a new word document and save it to disk.

import docx

# create a new document object

doc = docx.Document()

# add content to the document

doc.add\_heading('Document Title', level=1)

doc.add\_paragraph('This is the first paragraph.')

# save the document to disk

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

In this example, we first imported the “docx’ module, which provides the functionality for working with word documents. Then we create a new “document” object using the “docx.document()” constructor. We can then add the content to the document using the various methods provided by the “document” class, such as “add\_heading()”, and “add\_paragraph()”. Finally we save the document to the disk using the “save()” method and passing in the desired filename.

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 new paragraph with the text ‘Hello, There!’ to a document object stored in a variable named ‘doc’ in python, you can use the ‘add\_paragraph()’ method of the “docx.document” class. Here’s an example:-

import docx

# create a new document object

doc = docx.Document()

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

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

# save the document to disk

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

In this example, we first import the “docx” module and create a new ‘document’ object. Then we add a new paragraph with a desired text using the “add\_paragraph()” method of the “document” object. Finally, we save the document to the disk using the “save()” method.

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

**Ans:-** In word documents, there are nine levels of headings available, numbered 1 through 9. In python, using the “python.docx” module, you can specify the level of heading by passing an integer between 1 and 9 as the second argument to the “add\_heading()” method. Here’s an example:-

import docx

# create a new document object

doc = docx.Document()

# add a level-1 heading

doc.add\_heading('Heading Text', level=1)

# save the document to disk

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

In this example, We create a new “document” object and add a level-1 heading using the “add-heading()” method with the “level” argument set to 1. You can adjust the level of a heading by changing the integer value passed to the “level” argument.