Solutions

1. When using the PdfFileReader() and PdfFileWriter() functions from the PyPDF2 library to read from and write to PDF files, the modes in which the file objects should be opened are as follows:

   PdfFileReader(): The file object should be opened in binary mode using the 'rb' flag, which stands for read binary. This is because PDF files contain binary data that cannot be handled by text mode.

   PdfFileWriter(): The file object should be opened in binary mode using the 'wb' flag, which stands for write binary. This is because the PyPDF2 library writes binary data to the PDF file.

2.

```
from PyPDF2 import PdfFileReader

# Open the PDF file in read-binary mode
with open('example.pdf', 'rb') as f:
    # Create a PdfFileReader object to read the file
    pdf_reader = PdfFileReader(f)

    # Get the Page object for page 5
    page = pdf_reader.getPage(4)  # Note: pages are zero-indexed

    # Do something with the page object
    print(page.extractText())
```

3. In the PyPDF2 library for Python, the number of pages in a PDF document can be obtained using the getNumPages() method of the PdfFileReader class.

4. If a PDF file is encrypted with a password, you need to provide the password to PyPDF2 in order to decrypt the file and obtain Page objects from it.

   To do this, you can use the decrypt() method of the PdfFileReader class to provide the password. Here's an example:

```
from PyPDF2 import PdfFileReader

with open('encrypted.pdf', 'rb') as f:
    pdf_reader = PdfFileReader(f)

    # Check if the PDF file is encrypted
    if pdf_reader.isEncrypted:
        # Provide the password to decrypt the file
        pdf_reader.decrypt('swordfish')

    # Get the Page object for page 1
    page = pdf_reader.getPage(0)

    # Do something with the page object
    print(page.extractText())
```

5. To rotate a page in PyPDF2, you can use the rotateClockwise() or rotateCounterClockwise() methods of the PageObject class. These methods rotate the page 90 degrees clockwise or counterclockwise, respectively.

6.

7. In Python, you can use the python-docx library to work with Word documents. To obtain a list of Paragraph objects for a Document object that's stored in a variable named doc, you can use the paragraphs attribute of the Document object.

Here's an example:

```python
import docx

# Open the Word document
doc = docx.Document('example.docx')

# Get the list of Paragraph objects
paragraphs = doc.paragraphs

# Print the text of each paragraph
for paragraph in paragraphs:
    print(paragraph.text)
```

8. In the python-docx library for working with Word documents, the Run object has the bold, underline, italic, strike, and outline variables.

A Run object represents a contiguous run of text within a paragraph that has a consistent set of character formatting. Each Run object has its own set of character formatting, such as font family, font size, bold, italic, underline, strike, outline, and color. The formatting of a Run object can be modified independently of other runs within the same paragraph.

9.
In the python-docx library for working with Word documents, the bold variable of a Run object can have three possible values: True, False, or None.

If bold is set to True, the text in the Run object is displayed in bold font.
If bold is set to False, the text in the Run object is displayed in normal (non-bold) font.
If bold is set to None (the default value), the text in the Run object inherits the bold setting from the parent paragraph or style.

10.

```python
import docx

# Create a new Word document
doc = docx.Document()

# Add some text to the document
doc.add_paragraph('This is the first paragraph.')
doc.add_paragraph('This is the second paragraph.')

# Save the document to a file
doc.save('new_document.docx')
```

11.

```python
import docx

# Open an existing Word document
doc = docx.Document('existing_document.docx')

# Add a new paragraph with the text 'Hello, there!'
doc.add_paragraph('Hello, there!')

# Save the modified document to a new file
doc.save('modified_document.docx')
```

12.

In Word documents, there are generally six levels of headings available, each represented by an integer value:

Level 1 heading: represented by the integer 0

Level 2 heading: represented by the integer 1
Level 3 heading: represented by the integer 2
Level 4 heading: represented by the integer 3
Level 5 heading: represented by the integer 4
Level 6 heading: represented by the integer 5