Что такое python-docx и зачем она нужна

Python-docx — это мощная библиотека для работы с документами Microsoft Word в формате .docx. Она позволяет программно создавать, редактировать, читать и форматировать Word-документы без необходимости установки Microsoft Office. Библиотека особенно полезна для автоматизации создания отчетов, обработки больших объемов документов и интеграции с другими системами.

Установка и настройка python-docx

Для начала работы с библиотекой необходимо установить её через pip:

pip install python-docx

После установки можно сразу приступать к работе с документами Word в Python.

Создание нового документа Word

Создание нового документа — одна из базовых операций при работе с python-docx:

```
from docx import Document

# Создание нового документа
doc = Document()

# Добавление заголовков разных уровней
doc.add_heading('Главный заголовок', level=1)
doc.add_paragraph('Это основной текст документа с важной информацией.')

# Добавление подзаголовка
doc.add_heading('Подзаголовок', level=2)
doc.add_paragraph('Дополнительный текст под подзаголовком.')

# Сохранение документа с указанием пути
doc.save('new_document.docx')
print("Документ успешно создан!")
```

Чтение и извлечение данных из документа Word

Для работы с существующими документами используется следующий подход:

```
from docx import Document

# Открытие существующего документа
doc = Document('new_document.docx')

# Извлечение всего текста из документа
print("Содержимое документа:")
for paragraph in doc.paragraphs:
    if paragraph.text.strip():  # Пропускаем пустые параграфы
        print(f"- {paragraph.text}")

# Получение информации о заголовках
print("\nЗаголовки в документе:")
for paragraph in doc.paragraphs:
    if paragraph.style.name.startswith('Heading'):
```

```
        print(f"Заголовок уровня {paragraph.style.name[-1]}:
{paragraph.text}")
```

Работа с таблицами в Word документах

Python-docx предоставляет удобные инструменты для создания и заполнения таблиц:

```
from docx import Document
from docx.shared import Inches

# Создание документа с таблицей
doc = Document()
doc.add_heading('Отчет по продажам', level=1)

# Создание таблицы 4x4
table = doc.add_table(rows=4, cols=4)
table.style = 'Table Grid'  # Применение стиля таблицы

# Заполнение заголовков таблицы
headers = ['Товар', 'Количество', 'Цена', 'Сумма']
for i, header in enumerate(headers):
    table.cell(0, i).text = header
    # Делаем заголовки жирными
    table.cell(0, i).paragraphs[0].runs[0].bold = True

# Заполнение данных таблицы
data = [
    ['Яблоки', '10 кг', '150 руб/кг', '1500 руб'],
    ['Груши', '5 кг', '200 руб/кг', '1000 руб'],
    ['Апельсины', '8 кг', '180 руб/кг', '1440 руб']
]

for row_idx, row_data in enumerate(data, 1):
    for col_idx, cell_data in enumerate(row_data):
        table.cell(row_idx, col_idx).text = cell_data

# Сохранение документа
doc.save('sales_report.docx')
```

Добавление изображений в документ

Библиотека позволяет легко вставлять изображения с настройкой их размеров:

```
from docx import Document
from docx.shared import Inches, Cm

# Создание документа с изображением
doc = Document()
doc.add_heading('Отчет с изображениями', level=1)

# Добавление текста перед изображением
doc.add_paragraph('Ниже представлена диаграмма результатов:')

# Добавление изображения с указанием размеров
try:
    # Добавление изображения с заданной шириной
    doc.add_picture('chart.png', width=Inches(6.0))
```

```
    # Можно также использовать сантиметры
    doc.add_paragraph('Дополнительная схема:')
    doc.add_picture('scheme.jpg', width=Cm(10))

except FileNotFoundError:
    doc.add_paragraph('Изображение не найдено. Проверьте путь к файлу.')

# Сохранение документа
doc.save('document_with_images.docx')
```

Продвинутое форматирование текста

Python-docx предоставляет широкие возможности для форматирования текста:

```
from docx import Document
from docx.shared import Pt, RGBColor
from docx.enum.text import WD_COLOR_INDEX, WD_ALIGN_PARAGRAPH

# Создание документа с форматированием
doc = Document()

# Добавление заголовка с выравниванием по центру
title = doc.add_heading('Форматированный документ', level=1)
title.alignment = WD_ALIGN_PARAGRAPH.CENTER

# Создание параграфа с различным форматированием
p = doc.add_paragraph()

# Обычный текст
run1 = p.add_run('Обычный текст, ')

# Жирный текст
run2 = p.add_run('жирный текст, ')
run2.bold = True

# Курсив
run3 = p.add_run('курсив, ')
run3.italic = True

# Подчеркнутый текст
run4 = p.add_run('подчеркнутый, ')
run4.underline = True

# Цветной текст
run5 = p.add_run('цветной текст')
run5.font.color.rgb = RGBColor(255, 0, 0)  # Красный цвет

# Настройка шрифта и размера
p2 = doc.add_paragraph()
run6 = p2.add_run('Текст шрифтом Arial, размер 14pt')
run6.font.name = 'Arial'
run6.font.size = Pt(14)

# Выделение маркером
run7 = p2.add_run(' с выделением маркером')
run7.font.highlight_color = WD_COLOR_INDEX.YELLOW
```

```
# Сохранение документа
doc.save('formatted_document.docx')
```

Работа с метаданными документа

Python-docx позволяет управлять свойствами документа:

```
from docx import Document
from datetime import datetime

# Создание документа с метаданными
doc = Document()

# Установка свойств документа
doc.core_properties.title = 'Мой отчет'
doc.core_properties.author = 'Иван Петров'
doc.core_properties.subject = 'Ежемесячный отчет'
doc.core_properties.comments = 'Создано с помощью python-docx'
doc.core_properties.created = datetime.now()

# Добавление содержимого
doc.add_heading('Отчет за текущий месяц', level=1)
doc.add_paragraph('Основное содержимое отчета...')

# Сохранение
doc.save('report_with_metadata.docx')
```

Обработка ошибок и лучшие практики

При работе с python-docx важно учитывать возможные ошибки:

```
from docx import Document
import os


def safe_document_processing(file_path):
    """
    Безопасная обработка Word документа с обработкой ошибок
    """
    try:
        # Проверка существования файла
        if not os.path.exists(file_path):
            print(f"Файл {file_path} не найден")
            return None

        # Открытие документа
        doc = Document(file_path)

        # Извлечение текста
        full_text = []
        for paragraph in doc.paragraphs:
            if paragraph.text.strip():
                full_text.append(paragraph.text)

        return full_text

    except Exception as e:
```

```
        print(f"Ошибка при обработке документа: {e}")
        return None


# Использование функции
text_content = safe_document_processing('example.docx')
if text_content:
    print("Документ успешно обработан")
    for line in text_content:
        print(f"- {line}")
```

Заключение

Библиотека python-docx предоставляет мощные инструменты для автоматизации работы с документами Word. Она позволяет создавать профессионально оформленные документы, обрабатывать большие объемы данных и интегрировать Word-документы в рабочие процессы Python-приложений. Основные возможности включают создание и редактирование документов, работу с таблицами и изображениями, форматирование текста и управление метаданными.