Создание/изменение документов MS Word на Python

Сообщить об ошибке.

1. Сторонние пакеты и модули Python3.
2. Создание/изменение документов MS Word на Python

Модуль `python-docx` предназначен для создания и обновления файлов с расширением `.docx` - Microsoft Word. Этот модуль имеет одну зависимость: сторонний модуль `lxml`.

Установка модуля `python-docx` в виртуальное окружение.

Модуль `python-docx` размещен на PyPI, поэтому установка относительно проста.

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# ставим модуль python-docx
(VirtualEnv):~$ python3 -m pip install -U python-docx
```

Основы работы с файлами Microsoft Word на Python.

Открытие/создание документа.

Первое, что вам понадобится, это документ, над которым вы будете работать. Самый простой способ:

```
from docx import Document

# создание документа
document = Document()
# открытие документа
document = Document('/path/to/document.docx')
```

При этом создается пустой документ, основанный на "*шаблоне*" по умолчанию. Другими словами, происходит примерно то же самое, когда пользователь нажимает на иконку в Microsoft Word "*Новый документ*" с использованием встроенных значений по умолчанию.

При этом шрифт документа и его размер по умолчанию для всего документа можно задать следующим образом:

```
from docx import Document
from docx.shared import Pt

doc = Document()
# задаем стиль текста по умолчанию
style = doc.styles['Normal']
# название шрифта
style.font.name = 'Arial'
# размер шрифта
style.font.size = Pt(14)
document.add_paragraph('Текст документа')
```

Так же, можно открывать существующий документ Word и работать с ним при помощи модуля `python-docx`. Для этого, в конструктор класса `Document()` необходимо передать путь к существующему документу Microsoft Word.

Добавление заголовка документа.

В любом документе, основной текст делится на разделы, каждый из которых начинается с заголовка. Название таких разделов можно добавить методом `Document.add_heading()`:

```
# без указания аргумента `level`
# добавляется заголовок "Heading 1"
head = document.add_heading('Основы работы с файлами Microsoft Word на Python.')
from docx.enum.text import WD_ALIGN_PARAGRAPH
# выравнивание посередине
head.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

По умолчанию, добавляется заголовок верхнего уровня, который отображается в Word как "Heading 1". Если нужен заголовок для подраздела, то просто указываем желаемый уровень в виде целого числа от 1 до 9:

```
document.add_heading('Добавление заголовка документа', level=2)
```

Если указать `level=0`, то будет добавлен текст с встроенным стилем титульной страницы. Такой стиль может быть полезен для заголовка относительно короткого документа без отдельной титульной страницы.

Так же, заголовки разделов можно добавлять методом `document.add_paragraph().add_run()`, с указанным размером шрифта.

Например:

```
from docx import Document
from docx.shared import Pt

doc = Document()
# добавляем текст прогоном
run = doc.add_paragraph().add_run('Заголовок, размером 24 pt.')
# размер шрифта
run.font.size = Pt(24)
run.bold = True
doc.save('test.docx')
```

Добавление абзаца.

Абзацы в Word имеют основополагающее значение. Они используются для добавления колонтитулов, основного текста, заголовков, элементов списков, картинок и т.д.

Смотрим самый простой способ добавить абзац/параграф:

```
p = document.add_paragraph('Абзацы в Word имеют основополагающее значение.')
```

Метод `Document.add_paragraph()` возвращает ссылку на только что добавленный абзац (объект `Paragraph`). Абзац добавляется в конец документа. Эту ссылку можно использовать в качестве своеобразного "*курсора*" и например, вставить новый абзац прямо над ним:

```
prior_p = p.insert_paragraph_before(
          'Объект `paragraph` - это ссылка на только что добавленный абзац.')
```

Такое поведение позволяет вставить абзац в середину документа, это важно при изменении существующего документа, а не при его создании с нуля.

Ссылка на абзац, так же используется для его форматирования встроенными в MS Word стилями или для кастомного/пользовательского форматирования.

Пользовательское форматирование абзаца.

Форматирование абзацев происходит при помощи объекта `ParagraphFormat`.

Простой способ форматировать абзац/параграф:

```
from docx import Document
from docx.shared import Mm
from docx.enum.text import WD_ALIGN_PARAGRAPH

doc = Document()
# Добавляем абзац
p = doc.add_paragraph('Новый абзац с отступами и красной строкой.')
# выравниваем текст абзаца
p.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
```

```
# получаем объект форматирования
fmt = p.paragraph_format
# Форматируем:
# добавляем отступ слева
fmt.first_line_indent = Mm(15)
# добавляем отступ до
fmt.space_before = Mm(20)
# добавляем отступ слева
fmt.space_after = Mm(10)
doc.add_paragraph('Новый абзац.')
doc.add_paragraph('Еще новый абзац.')
doc.save('test.docx')
```

Чтобы узнать, какие параметры абзаца еще можно настроить/изменить, смотрите материал "Объект `ParagraphFormat`"

Очень часто в коде, с возвращенной ссылкой (в данном случае `p`) ничего делать не надо, следовательно нет смысла ее присваивать переменной.

Применение встроенного стиля в Microsoft Word к абзацу.

Стиль абзаца - это набор правил форматирования, который заранее определен в Microsoft Word, и храниться в редакторе в качестве переменной. По сути, стиль позволяет сразу применить к абзацу целый набор параметров форматирования.

Можно применить стиль абзаца, прямо при его создании:

```
document.add_paragraph('Стиль абзаца как цитата', style='Intense Quote')
document.add_paragraph('Стиль абзаца как список.', style='List Bullet')
```

В конкретном стиле `'List Bullet'`, абзац отображается в виде маркера. Также можно применить стиль позже. Две строки, в коде ниже, эквивалентны примеру выше:

```
document.add_paragraph('Другой стиль абзаца.').style = 'List Number'

# Эквивалентно

paragraph = document.add_paragraph('Другой стиль абзаца.')
# применяем стиль позже
paragraph.style = 'List Number'
```

Стиль указывается с использованием его имени, в этом примере имя стиля - `'List'`. Как правило, **имя стиля** точно такое, как оно отображается в пользовательском интерфейсе Word.

Обратите внимание, что можно установить встроенный стиль прямо на результат `document.add_paragraph()`, без использования возвращаемого объекта `paragraph`

Жирный, курсив и подчеркнутый текст в абзаце.

Разберемся, что происходит внутри абзаца:

- Абзац содержит все форматирование на уровне блока, такое как - отступ, высота строки, табуляции и так далее.
- Форматирование на уровне символов, например полужирный и курсив, применяется на уровне прогона `paragraph.add_run()`. Все содержимое абзаца должно находиться в пределах цикла, но их может быть больше одного. Таким образом, для абзаца с полужирным словом посередине требуется три прогона: обычный, полужирный - содержащий слово, и еще один нормальный для текста после него.

Когда создается абзац методом `Document.add_paragraph()`, то передаваемый текст добавляется за один прогон `Run`. Пустой абзац/параграф можно создать, вызвав этот метод без аргументов. В этом случае, наполнить абзац текстом можно с помощью метода `Paragraph.add_run()`. Метод абзаца `.add_run()` можно вызывать несколько раз, тем самым добавляя информацию в конец данного абзаца:

```
paragraph = document.add_paragraph('Абзац содержит форматирование ')
paragraph.add_run('на уровне блока.')
```

В результате получается абзац, который выглядит так же, как абзац, созданный из одной строки. Если не смотреть на полученный XML, то не очевидно, где текст абзаца разбивается на части. Обратите внимание на конечный пробел в конце первой строки. Необходимо четко указывать, где появляются пробелы в начале и в конце прогона, иначе текст будет слитный (без пробелов). Они (пробелы) автоматически не вставляются между прогонами `paragraph.add_run()`. Метод `paragraph.add_run()` возвращает ссылку на объект прогона `Run`, которую можно использовать, если она нужна.

Объекты прогонов имеют следующие свойства, которые позволяют установить соответствующий стиль:

- `.bold`: полужирный текст;
- `.underline`: подчеркнутый текст;
- `.italic`: курсивный (наклонный) текст;
- `.strike`: зачеркнутый текст.

```
paragraph = document.add_paragraph('Абзац содержит ')
paragraph.add_run('форматирование').bold = True
paragraph.add_run(' на уровне блока.')
```

Получится текст, что то вроде этого: "Абзац содержит **форматирование** на уровне блока".

Обратите внимание, что можно установить полужирный или курсив прямо на результат `paragraph.add_run()`, без использования возвращаемого объекта прогона:

```
paragraph.add_run('форматирование').bold = True
# или
run = paragraph.add_run('форматирование')
run.bold = True
```

Передавать текст в метод `Document.add_paragraph()` не обязательно. Это может упростить код, если строить абзац из прогонов:

```
paragraph = document.add_paragraph()
paragraph.add_run('Абзац содержит ')
paragraph.add_run('форматирование').bold = True
paragraph.add_run(' на уровне блока.')
```

Пользовательское задание шрифта прогона.

```
from docx import Document
from docx.shared import Pt, RGBColor

# создание документа
doc = Document()
# добавляем текст прогоном
run = doc.add_paragraph().add_run('Заголовок, размером 24 pt.')
# название шрифта
run.font.name = 'Arial'
# размер шрифта
run.font.size = Pt(24)
# цвет текста
run.font.color.rgb = RGBColor(0, 0, 255)
# + жирный и подчеркнутый
run.font.bold = True
run.font.underline = True
doc.save('test.docx')
```

Применение стилей Microsoft Word к символам текста (к прогону).

В дополнение к встроенным стилям абзаца, которые определяют группу параметров уровня абзаца, Microsoft Word имеет стили символов, которые определяют группу параметров уровня прогона `paragraph.add_run()`. Другими словами, можно думать о стиле текста как об указании шрифта, включая его имя, размер, цвет, полужирный, курсив и т. д.

Подобно стилям абзацев, стиль символов текста будет определен в документе, который открывается с помощью вызова `Document()` (см. Общие сведения о стилях).

Стиль символов можно указать при добавлении нового прогона:

```
paragraph = document.add_paragraph('Обычный текст, ')
paragraph.add_run('текст с акцентом.', 'Emphasis')
```

Также можно применить стиль к прогону после его добавления. Этот код дает тот же результат, что и строки выше:

```
paragraph = document.add_paragraph()
paragraph.add_run('Обычный текст, ')
paragraph.add_run('текст с акцентом.').style = 'Emphasis'
```

Как и в случае со стилем абзаца, имя стиля текста такое, как оно отображается в пользовательском интерфейсе Word.

Пользовательский стиль символов текста.

```
from docx import Document
from docx.shared import Pt, RGBColor

# создание документа
doc = Document()
# задаем стиль текста по умолчанию
style = doc.styles['Normal']
# название шрифта
style.font.name = 'Calibri'
# размер шрифта
style.font.size = Pt(14)
p = doc.add_paragraph('Пользовательское ')
# добавляем текст прогоном
run = p.add_run('форматирование ')
# размер шрифта
run.font.size = Pt(16)
# курсив
run.font.italic = True
# добавляем еще текст прогоном
run = p.add_run('символов текста.')
# Форматируем:
# название шрифта
run.font.name = 'Arial'
# размер шрифта
run.font.size = Pt(18)
# цвет текста
run.font.color.rgb = RGBColor(255, 0, 0)
# + жирный и подчеркнутый
run.font.bold = True
run.font.underline = True
doc.save('test.docx')
```

Добавление разрыва страницы.

При создании документа, время от времени нужно, чтобы следующий текст выводился на отдельной странице, даже если последняя не заполнена. Жесткий разрыв страницы можно сделать следующим образом:

```
document.add_page_break()
```

Если вы обнаружите, что используете это очень часто, это, вероятно, знак того, что вы могли бы извлечь выгоду, лучше разбираясь в стилях абзацев. Одно свойство стиля абзаца, которое можно установить, - это разрыв страницы непосредственно перед каждым абзацем, имеющим этот стиль. Таким образом, можно установить заголовки определенного уровня, чтобы всегда начинать новую страницу. Подробнее о стилях позже. Они оказываются критически важными для получения максимальной отдачи от Word.

Жесткий разрыв страницы можно привязать к стилю абзаца, и затем применять его для определенных абзацев, которые должны начинаться с новой страницы. Так же можно установить жесткий разрыв на стиль заголовка определенного уровня, чтобы с него всегда начинать новую страницу. В общем, стили, оказываются критически важными для того, чтобы получить максимальную отдачу от модуля `python-docx`.

Добавление картинки в документ.

Microsoft Word позволяет разместить изображение в документе с помощью пункта меню "*Вставить изображение*". Вот как это сделать при помощи модуля `python-docx`:

```
document.add_picture('/path/to/image-filename.png')
```

В этом примере используется путь, по которому файл изображения загружается из локальной файловой системы. В качестве пути можно использовать файловый объект, по сути, любой объект, который действует как открытый файл. Такое поведение может быть полезно, если изображение извлекается из базы данных или передается по сети.

Размер изображения.

По умолчанию, изображение добавляется с исходными размерами, что часто не устраивает пользователя. Собственный размер рассчитывается как `px/dpi`. Таким образом, изображение размером 300x300 пикселей с разрешением 300 точек на дюйм появляется в квадрате размером один дюйм. Проблема в том, что большинство изображений не содержат свойства `dpi`, и по умолчанию оно равно 72 dpi. Следовательно, то же изображение будет иметь одну сторону, размером 4,167 дюйма, что означает половину страницы.

Чтобы получить изображение нужного размера, необходимо указывать его ширину или высоту в удобных единицах измерения, например, в миллиметрах или сантиметрах:

```
from docx.shared import Mm

document.add_picture('/path/to/image-filename.png', width=Mm(35))
```

Если указать только одну из сторон, то модуль `python-docx` использует его для вычисления правильно масштабированного значения другой стороны изображения. Таким образом сохраняется соотношение сторон и изображение не выглядит растянутым.

Классы `Mm()` и `Cm()` предназначены для того, чтобы можно было указывать размеры в удобных единицах. Внутри `python-docx` используются английские метрические единицы, 914400 дюймов. Так что, если просто указать размер, что-то вроде `width=2`, то получится очень маленькое изображение. Классы `Mm()` и `Cm()` импортируются из подпакета `docx.shared`. Эти классы можно использовать в арифметике, как если бы они были целыми числами. Так что выражение, width=Mm(38)/thing_count, работает нормально.

Чтение документов Microsoft Word.

В модуле `python-docx`, структура документа Microsoft Word представлена тремя различными типами данных. На самом верхнем уровне объект `Document()` представляет собой весь документ. Объект `Document()` содержит список объектов `Paragraph()`, которые представляют собой абзацы документа. Каждый из абзацев содержит список, состоящий из одного или нескольких объектов `Run()`, представляющих собой фрагменты текста с различными стилями форматирования.

Например:

```
>>> from docx import Document
>>> doc = Document('/path/to/example.docx')
# количество абзацев в документе
>>> len(doc.paragraphs)
# текст первого абзаца в документе
>>> doc.paragraphs[0].text
# текст второго абзаца в документе
>>> doc.paragraphs[1].text
# текст первого прогона второго абзаца
>>> doc.paragraphs[1].runs[0].text
```

Используя следующий код, можно получить весь текст документа:

```
text = []
for paragraph in doc.paragraphs:
    text.append(paragraph.text)
print('\n\n'.join(text))
```

А так можно получить стили всех параграфов:

```
styles = []
for paragraph in doc.paragraphs:
    styles.append(paragraph.style)
```

Использовать полученные стили можно следующим образом:

```
# изменим стиль 1 параграфа на
# стиль взятый из 3 параграфа
doc.paragraphs[0].style = styles[2]
```