Работа с документами

Довольно часто при разработке возникает необходимость программно создать какой-то документ или внести изменения в уже существующий. Работая с текстами разной направленности из кода, нужно принять во внимание, что тексты иногда хранятся в более сложных форматах, чем .txt. Они могут содержать встроенное форматирование, быть разделенными на страницы, перемежаться медиаконтентом (графиками, рисунками).

Python умеет работать со многими такими документами. Давайте посмотрим, что можно сделать, чтобы создавать документы в формате Word, Excel или PowerPoint прямо из Python.

Стоит отметить, что форматы .docx, .xlsx и .pptx открытые, что позволяет разработчикам довольно просто писать библиотеки для работы с ними. Для каждого офисного формата есть несколько библиотек с разным функционалом, и мы рассмотрим лишь некоторые из них.

Модуль docx

Создание, открытие и сохранение документа

Давайте воспользуемся модулем python-docx для создания docx-документа. Он, как и остальные приведенные в данном уроке библиотеки, не входит в состав стандартной библиотеки и требует отдельной установки через pip.

```
pip install python-docx
```

После установки давайте рассмотрим вот такой пример:

```
from docx import Document

from docx.shared import Inches


document = Document()


document.add_heading('Заголовок документа', 0)


document.add_paragraph('Абзац без форматирования')


# тут у нас будет более сложный абзац
```

```
p = document.add_paragraph('Часть абзаца обычным текстом,
')

p.add_run('часть жирным шрифтом, ').bold = True

p.add_run(' а часть ')

p.add_run('курсивом.').italic = True


document.add_heading('Заголовок первого уровня', level=1)

document.add_paragraph('Некоторая цитата', style='Intense
Quote')


document.add_paragraph('Элемент ненумерованного списка',

                       style='List Bullet')

document.add_paragraph('Элемент нумерованного списка',

                       style='List Number')


table = document.add_table(rows=1, cols=3)

hdr_cells = table.rows[0].cells

hdr_cells[0].text = 'Номер'

hdr_cells[1].text = 'Название'

hdr_cells[2].text = 'Количество'


document.save('test.docx')
```

Ключевой элемент этой библиотеки — объект `Document`. Для создания нового документа формата docx надо записать результат вызова `Document` в переменную. Этого достаточно для создания абсолютно пустого документа в памяти. Чтобы наполнить его содержимым, необходимо вызывать у получившегося объекта различные методы, например:

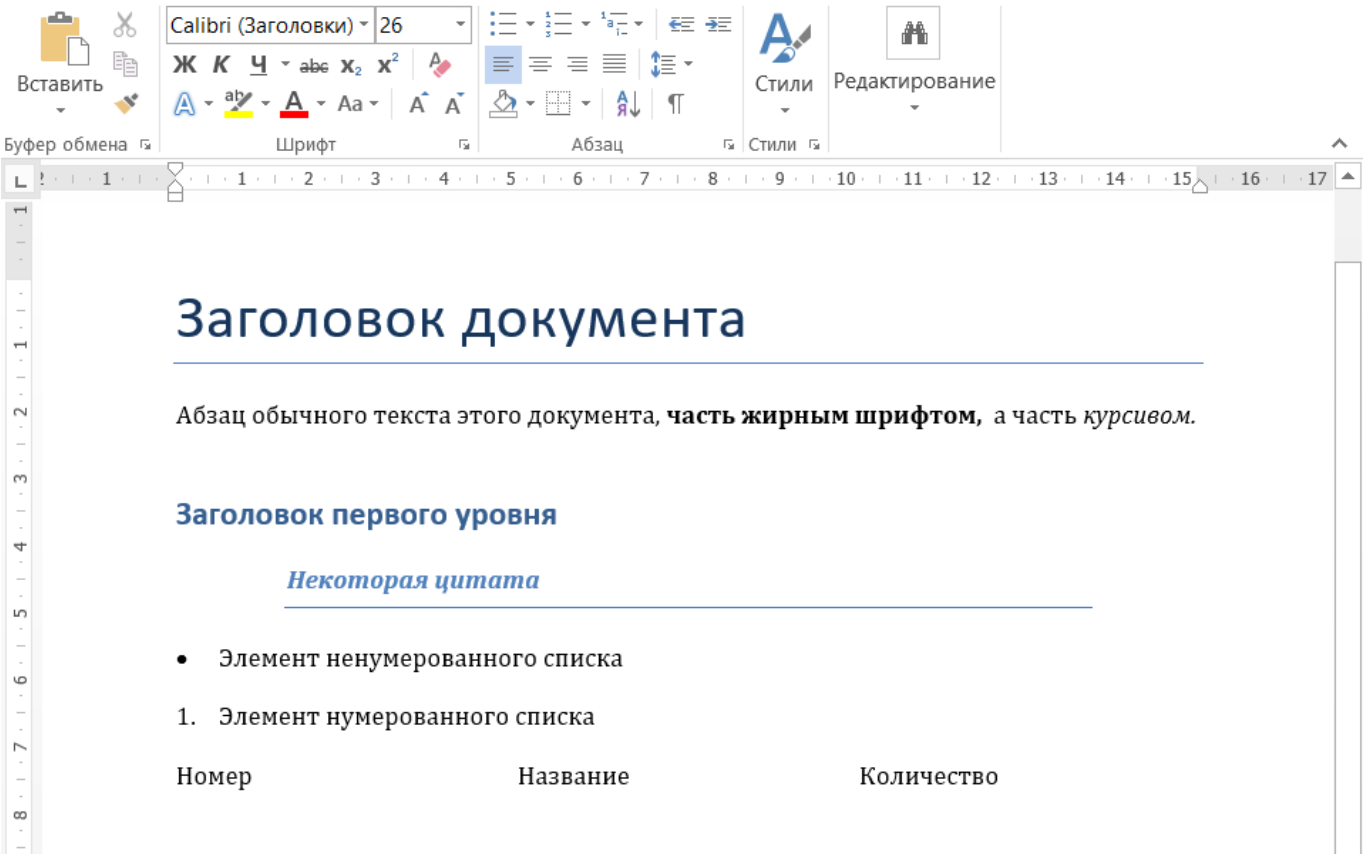
- `add_heading` — для создания заголовков разного уровня. При этом уровень 0 — заголовок документа
- `add_paragraph` — для создания абзацев. С помощью параметра `style` можно управлять стилем абзаца и превращать его элемент маркированного или нумерованного списка

Метод `add_paragraph` объекта-документа возвращает новый абзац, у которого есть свои методы вроде `add_run`, который дописывает текст с форматированием или без него в конец абзаца.

Метод `add_table` создает в документе таблицу размером `col` столбцов и `row` строк. Если мы присвоим результат вызова этой функции переменной, сможем заполнить такую таблицу данными.

Метод `save` сохраняет документ под указанным именем.

Выполните приведенный код. Затем откройте созданный файл test.docx. Вы должны увидеть что-то такое:



С помощью python-docx можно не только создавать новые документы, но и открывать уже существующие. Для этого при создании документа надо в качестве параметра передать путь к файлу или просто его имя, если он находится в каталоге с программой.

Таким образом, мы можем получить абзацы или другие элементы документа, пройтись по ним и получить либо изменить какую-то информацию:

```
from docx import Document


document = Document('test.docx')

for par in document.paragraphs:
```

```
    print(par.text)


for table in document.tables:
    print(table.rows[0].cells[0].text)
```

```
Заголовок документа

Абзац без форматирования

Часть абзаца обычным текстом, часть жирным шрифтом,  а часть
курсивом.

Заголовок первого уровня

Некоторая цитата

Элемент ненумерованного списка

Элемент нумерованного списка

Номер
```

Как мы увидим дальше, работа с подобными модулями примерно одинакова. Все элементы управления и форматирования:

- Списки
- Абзацы
- Таблицы
- Ячейки

есть в наличии в том или ином виде, их можно изменять и комбинировать.

Основные текстовые элементы и их форматирование

Абзац

Добавление абзаца:

```
paragraph = document.add_paragraph()
```

или так:

```
paragraph = document.add_paragraph('Вот дом, ')


paragraph.add_run('который построил Джек.')  # добавление
текста в абзац
```

Заголовки

Добавление заголовков:

```
document.add_heading('Сказки Матушки Гусыни')
```

По умолчанию создается заголовок первого уровня ('Heading 1'), но можно указать уровень:

```
document.add_heading('Стихи и песенки', level=2)
```

Если указать уровень 0, то будет применен стиль названия документа.

Разрыв страницы

Если нужно перейти на новую страницу, хотя предыдущая заполнена не до конца, можно вставить разрыв страницы:

```
document.add_page_break()
```

Таблицы

Вставить таблицу с заданным числом строк и столбцов можно так:

```
table = document.add_table(rows=2, cols=2)
```

Заполним созданную таблицу данными из списка, добавим заголовки столбцов и применим к ней стиль:

```
from docx import Document


document = Document()


table = document.add_table(4, 3)

heading = ("No", "Month", "Season")

head_row = table.rows[0].cells

for i in range(len(heading)):

    head_row[i].text = heading[i]

data = ((1, "January", "Winter"),

        (2, "February", "Winter"),

        (3, "March", "Spring"))

for i in range(len(data)):

    cells = table.rows[i + 1].cells
```

```
    for j in range(len(data[0])):
        cells[j].text = str(data[i][j])
table.style = 'Light Shading Accent 1'

document.save("example.docx")
```

Получим таблицу:

No	Month	Season
1	January	Winter
2	February	Winter
3	March	Spring

Вставка картинки

```
document.add_picture('python.png')
```

По умолчанию картинка вставляется с ее реальным размером, что часто занимает слишком большую часть страницы. Чтобы изменить размеры картинки, нужно указать желаемую величину ширины или высоты в дюймах или сантиметрах:

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

document = Document()
document.add_picture('python.png')
document.add_paragraph()
document.add_picture("python.png", width=Inches(1.0))
document.save("example.docx")
```

Результат:

Списки

Списки создаются, применяя стили к абзацам:

```
par = document.add_paragraph("Первый элемент
нумерованного списка")

# применяем стиль после добавления параграфа

par.style = 'List Number'

# добавляем параграф и сразу применяем стиль

par = document.add_paragraph('Второй элемент
нумерованного списка').style = 'List Number'

# добавляем параграф, стиль передаем как аргумент

document.add_paragraph('Маркированный список',
style='List Bullet')
```

```
document.add_paragraph('Второй уровень многоуровневого
списка', style='List Bullet 2')

document.save("example.docx")
```

Применение стилей

Стили в модуле python-docx представляют собой целые блоки характеристик, которые можно сформировать и применить к абзацу или прогону текста целиком. (Прогон – это часть текста, к которой может быть применено форматирование. Оно может отличаться от форматирования других прогонов, или частей текста.)

Форматирование символов

Форматирование символов происходит на уровне прогона (run). Даже если строка выглядит как единое целое, но в ней встречаются отдельные блоки с разным начертанием, значит, она создавалась из нескольких прогонов.

Можно изменить начертание символов, размер, тип шрифта. Для этого есть несколько способов:

```
paragraph = document.add_paragraph('This is the house
that ')

run = paragraph.add_run('Jack')

run.bold = True

paragraph.add_run(' built.')
```

или так:

```
paragraph = document.add_paragraph()

paragraph.add_run('This is the house that ')

paragraph.add_run('Jack').bold = True

paragraph.add_run(' built.')
```

Для изменения начертания на курсив, применим тип italic к прогону текста:

```
paragraph.add_run('Jack').italic = True
```

За размер и тип шрифта отвечают соответственно атрибуты name и size объекта Font:

```
from docx import Document



document = Document()
```

```
run = document.add_paragraph().add_run("This is the malt
that lay in the house that Jack built.")

font = run.font

font.name = 'Times New Roman'

font.size = Pt(16)
```

Также можно изменить цвет шрифта, используя, например, модуль shared из библиотеки docx:

```
from docx.shared import RGBColor



font.color.rgb = RGBColor(0x42, 0x24, 0xE9)
```

Форматирование абзацев

Задание стилей форматирования абзаца:

```
par = paragraph.paragraph_format
```

Выравнивание

```
WD_ALIGN_PARAGRAPH.LEFT - по левому краю;

WD_ALIGN_PARAGRAPH.CENTER  # по центру;

WD_ALIGN_PARAGRAPH.RIGHT  # по правому краю;

WD_ALIGN_PARAGRAPH.JUSTIFY  # по ширине
```

Пример использования

```
from docx.enum.text import WD_ALIGN_PARAGRAPH



document = Document()

paragraph = document.add_paragraph("This is the rat that
ate the malt")

paragraph_format = paragraph.paragraph_format

paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

Отступы

Отступом называется горизонтальный сдвиг текста относительно границ блока, в котором он расположен, обычно это границы области текста на странице.

Можно задать отступ для всего абзаца слева или справа:

```
from docx.shared import Inches, Mm



paragraph = document.add_paragraph("This is the dog that
worried the cat")

paragraph_format = paragraph.paragraph_format

paragraph_format.left_indent = Inches(0.5)

paragraph_format.right_indent = Pt(24)
```

или отступ первой строки:

```
paragraph_format.first_line_indent = Mm(-15)
```

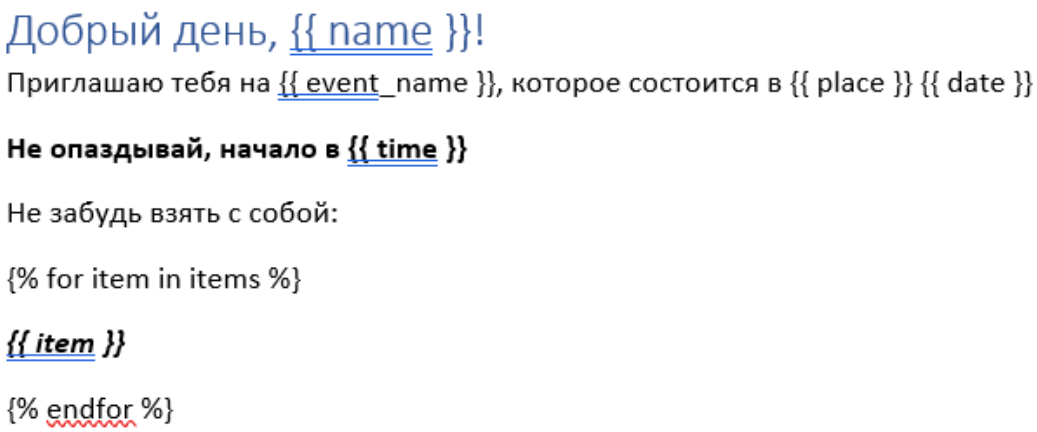
Мы рассмотрели только некоторые возможности модуля, полная документация находится здесь.

Работа с документом как с шаблоном

Очень распространенным вариантом использования является не создание документа с нуля, а заполнение данными уже готового шаблона. Для этого можно использовать библиотеку docxtpl.

```
pip install docxtpl
```

Например, у нас есть вот такой шаблон приглашения на мероприятие:



С помощью двойных фигурных скобок выделены места в документе, куда мы можем подставить свои данные по имени. Кроме того, шаблоны поддерживают специальный синтаксис для циклов, условий и других конструкций. Внутри docxtpl лежит мощный и простой в использовании

движок шаблонов jinja2. Вы с ним еще встретитесь на втором году обучения, когда мы будем говорить о веб-программировании.

```
from docxtpl import DocxTemplate

import datetime as dt



doc = DocxTemplate("tpl.docx")

context = {

    'name': 'Иван Иванович',

    'event_name': "танцевально-увеселительное мероприятие
по случаю 45-летия независимости Кабо-Верде",

    'place': "любой точке Кабо-Верде",

    'date': dt.date.today(),

    'time': dt.datetime.now().strftime("%H:%M"),

    'items': ["картину",

              'корзину',

              "картонку",

              "маленькую собачонку"]

}

doc.render(context)

doc.save("res.docx")
```

При работе с этой библиотекой нам надо воспользоваться классом `DocxTemplate`, при создании экземпляра которого необходимо передать имя нашего документа, который содержит шаблон. После чего нужно создать словарь и заполнить его информацией для вставки в документ. Значения из нашего словаря будут подставляться в шаблон по ключу. Метод `render` как раз и отвечает за такое заполнение. Метод `save` сохраняет документ под переданным именем.

В итоге у нас получится вот такой документ:

Добрый день, Иван Иванович!

Приглашаю тебя на танцевально-увеселительное мероприятие по случаю 45-летия независимости Кабо-Верде, которое состоится в любой точке Кабо-Верде 2019-06-25

Не опаздывай, начало в 15:59

Не забудь взять с собой:

картину

корзину

картонку

маленькую собачонку

Создание презентаций

Для работы с презентациями формата `pptx` в Python есть библиотека python-pptx.

```
pip install python-pptx
```

Принцип ее работы во многом схож с библиотекой python-docx, но здесь ключевым элементом, с которым идет работа, выступает не `Document`, а `Presentation`, который вызывается без аргументов, если нужно создать новую презентацию, или с именем файла существующей презентации, если нужно ее открыть для анализа или изменения.

Макеты

Презентация состоит из слайдов, на которых объекты могут быть расположены разными способами, в соответствии с выбранным макетом (`slide_layouts`). Макетов всего 9:

1. **Title** (титульный слайд презентации)
2. **Title and Content** (заголовок и содержимое)
3. **Section Header** (заголовок раздела)
4. **Two Content** (два текстовых блока для списков)
5. **Comparison** (сравнение – два текстовых блока с заголовками у каждого)
6. **Title Only** (только заголовок)
7. **Blank** (пустой слайд)
8. **Content with Caption** (содержимое и заголовок)

9. **Picture with Caption** (рисунок и заголовок)

Контейнеры

Для размещения на слайде объектов – текстовых блоков, списков, иллюстраций, таблиц, диаграмм – предусмотрены специальные контейнеры (placeholders), которые являются разновидностью более общего типа объектов shapes. Создадим презентацию с титульным слайдом, разместим на нем заголовок и подзаголовок:

```
from pptx import Presentation



# создаем новую презентацию

prs = Presentation()

# получаем схему расположения элементов для заголовочного
слайда

title_slide_layout = prs.slide_layouts[0]

# создаем титульный слайд

slide = prs.slides.add_slide(title_slide_layout)

# создаем у слайда заголовок и текст

title = slide.shapes.title

subtitle = slide.placeholders[1]



title.text = "Тестовый заголовок"

subtitle.text = "Тестовый подзаголовок"
```

Подготовим макет слайда для размещения на нем иллюстрации. Лучше всего подходит макет 8: у него есть специальный контейнер для размещения картинки (2). Также там есть placeholder для заголовка и для размещения текста. Картинку можно использовать любую, но для того, чтобы работал именно этот код важно, чтобы она имела такое же имя и лежала в папке с программой.

```
# создаем новый слайд со схемой для добавления
изображений

slide = prs.slides.add_slide(prs.slide_layouts[8])

slide.shapes.title.text = "А теперь с картинкой"

# добавляем изображение
```

```
placeholder = slide.placeholders[1]

placeholder.insert_picture('photo.jpg')
```

Форматирование символов

Для форматирования текста можно создать текстовое поле, добавить в него абзац, а затем в него добавлять прогоны с нужным нам форматированием. Все очень похоже на работу с символами в docx.

Но на нашем макете уже есть текстовый placeholder. Поскольку он является разновидностью объекта shape, у него есть атрибуты text для записи текста без форматирования и text_frame для форматированного текста, то есть новое текстовое поле создавать не нужно. Воспользуемся этим для форматирования текста. Импортируем необходимые функции для задания единиц измерения, выравнивания текста и применения цвета символа.

```
from pptx.dml.color import RGBColor

from pptx.util import Pt, Inches

from pptx.enum.text import PP_ALIGN



txt_placeholder = slide.placeholders[2]

txt_placeholder.text = "Пояснительный текст под картинкой
без форматирования"



t = txt_placeholder.text_frame

p = t.add_paragraph()

run = p.add_run()

run.text = "Пояснительный текст "

font = run.font

font.size = Pt(18)

font.name = "Time New Roman"



run1 = p.add_run()

run1.text = "под картинкой с форматированием"

run1.font.italic = True
```

```
run1.font.size = Pt(18)
```

Работа с пустым слайдом

Поэкспериментируем с пустым слайдом, на который можно добавить любые фреймы.

```
# добавим пустой слайд (макет 6)

empty = prs.slides.add_slide(prs.slide_layouts[6])



# добавим текстовое поле для текста с форматированием символов

left = top = Inches(1.0)

width = Inches(8)

height = Inches(1)

txt_box = empty.shapes.add_textbox(left, top, width, height)

txt_frame = txt_box.text_frame

p = txt_frame.add_paragraph()

run = p.add_run()

run.text = "Добавление объектов"

font = run.font

font.size = Pt(36)

# меняем цвет шрифта

font.color.rgb = RGBColor(0x0, 0x70, 0xf0)

# меняем выравнивание абзаца

p.alignment = PP_ALIGN.CENTER
```

Списки

В библиотеке pptx не возможности устанавливать маркеры списков самостоятельно, можно только добавить текст в placeholder соответствующего типа, но можно сымитировать список с помощью отступа level=1 и размещения маркера перед текстом:

```
# создаем и размещаем на слайде текстовое поле

left = Inches(1.5)
```

```
top = Inches(2)

width = Inches(4)

height = Inches(4)

txBox = empty.shapes.add_textbox(left, top, width,
height)

tf = txBox.text_frame

# добавляем абзац

p1 = tf.add_paragraph()

p1.text = "На слайд можно добавить:"

p1.font.size = Pt(24)

# и список

data = ["фигуру;", "картинку;", "таблицу;", "список..."]

for item in data:

    add = tf.add_paragraph()

    add.text = f"- {item}"

    add.level = 1
```

Гиперссылки

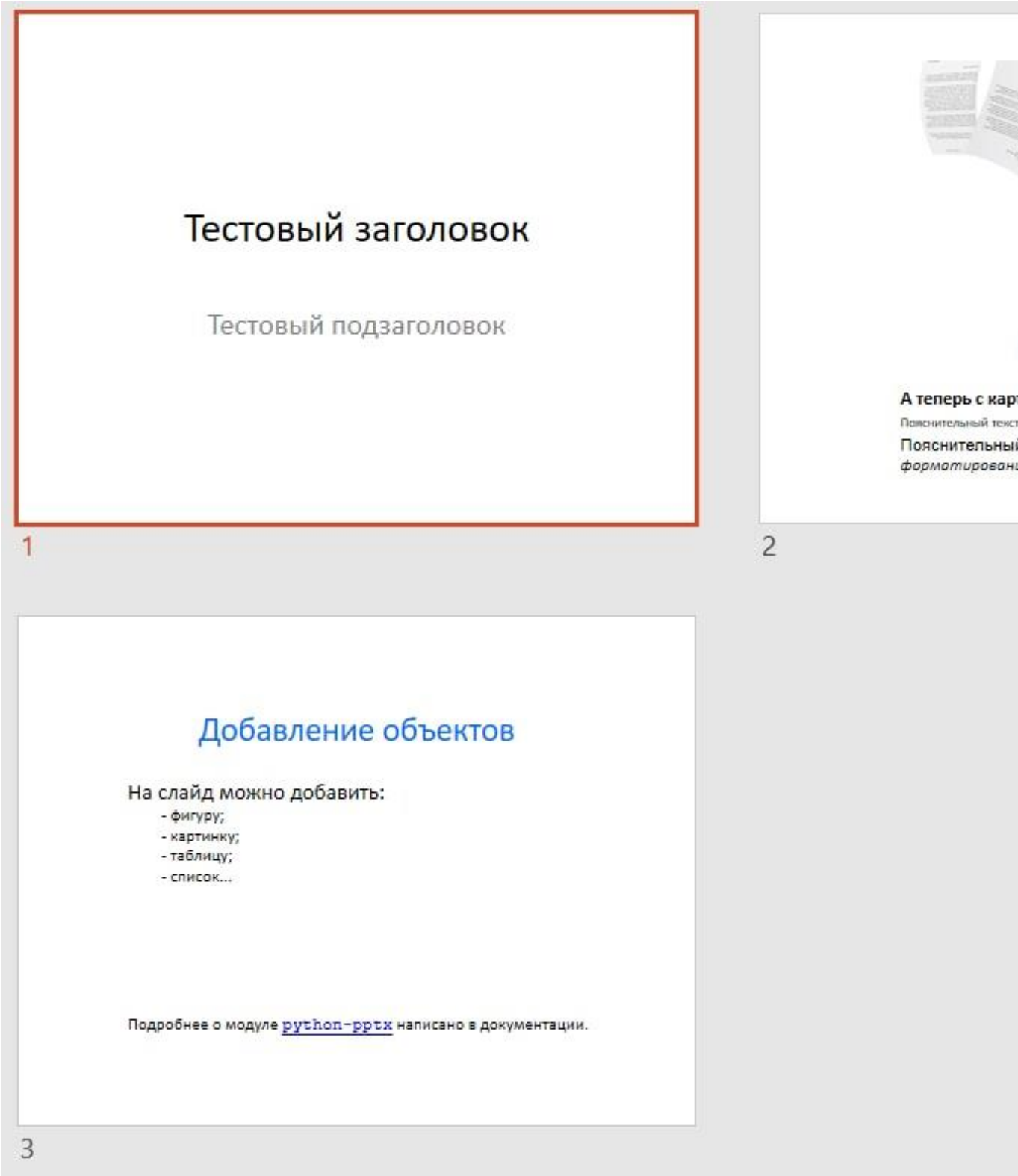
На последнем слайде добавим текстовый блок для размещения гиперссылки на сайт с документацией по этому модулю:

```
# Текстовый блок с гиперссылкой

left = Inches(1.5)

top = Inches(5.5)

width = Inches(8)

height = Inches(1)

txBox = empty.shapes.add_textbox(left, top, width,
height)

tf = txBox.text_frame

p2 = tf.add_paragraph()

run = p2.add_run()
```

```
run.text = "Подробнее о модуле "

run1 = p2.add_run()

run1.text = "python-pptx"

font = run1.font

font.name = "Courier New"

font.color.rgb = RGBColor(0xff, 0x00, 0x00)

run2 = p2.add_run()

run2.text = " написано в документации."



run1.hyperlink.address = "https://python-
pptx.readthedocs.io/en/latest/"

# сохраняем презентацию

prs.save('test.pptx')
```

Соберите программу из рассмотренных фрагментов, вынесите импорты в начало файла, не забудьте поместить картинку в папку с программой. Запустите и полюбуйтесь на результат.

Должна получиться вот такая презентация:



Как и в случае с документами, мы можем открыть уже существующую презентацию и получить или изменить какие-либо данные.

```
from pptx import Presentation


prs = Presentation('test.pptx')
```

```
slide = prs.slides[0]

print(slide.placeholders[0].text)
```

```
Тестовый заголовок
```

Создание таблиц Excel

Для работы с файлами формата .xlsx есть несколько библиотек, одна из них — xlsxwriter, предназначенная только для создания xlsx-файлов.

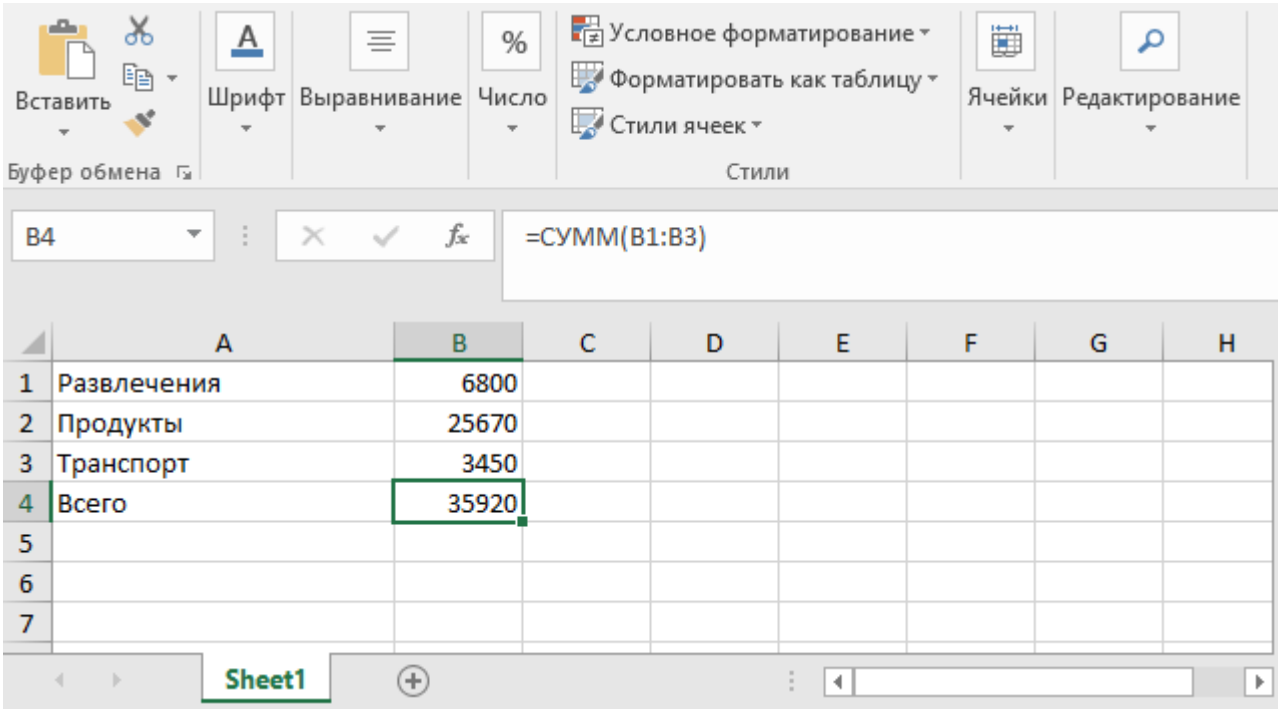
```
pip install xlsxwriter
```

Принцип работы с этой библиотекой очень схож с рассмотренными ранее, но с небольшими оговоркамии. Уже при создании документа нам необходимо передавать имя, под которым этот документ будет сохранен в конце.

```
import xlsxwriter


workbook = xlsxwriter.Workbook('Суммы.xlsx')

worksheet = workbook.add_worksheet()


data = [('Развлечения', 6800), ('Продукты', 25670),
('Транспорт', 3450),]


for row, (item, price) in enumerate(data):

    worksheet.write(row, 0, item)

    worksheet.write(row, 1, price)


row += 1

worksheet.write(row, 0, 'Всего')

worksheet.write(row, 1, '=SUM(B1:B3)')


workbook.close()
```

Потом с помощью метода `add_worksheet` надо добавить страницу в документ. У добавленной страницы можно вызывать метод `write`,

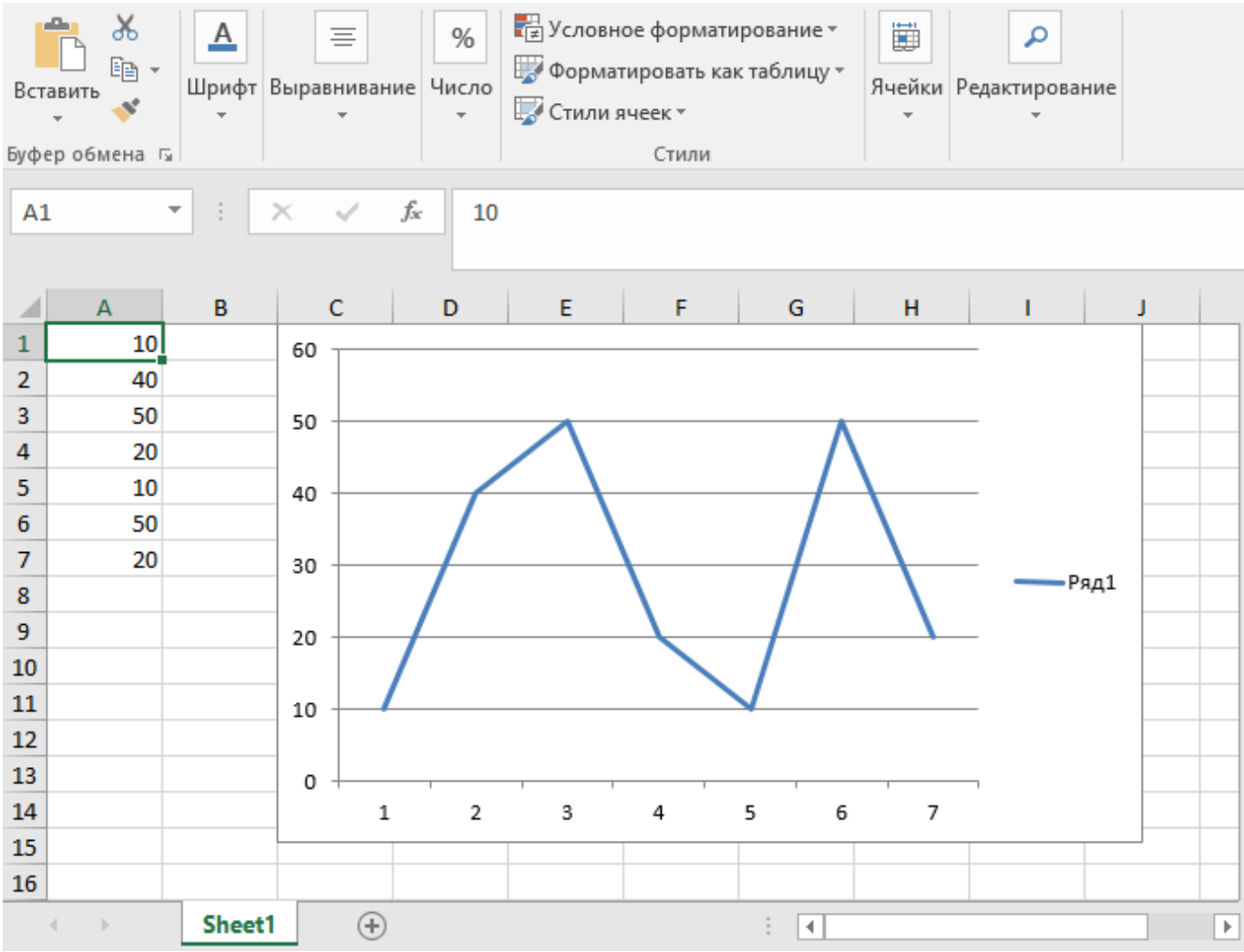
который записывает в определенную строку и колонку переданные данные.



С помощью метода `add_chart` можно создать диаграмму, указав ее тип и данные для построения. После чего построенную диаграмму можно добавить на страницу с помощью метода `insert_chart` с указанием ячейки вставки и самой диаграммы.

```
import xlsxwriter


workbook = xlsxwriter.Workbook('диаграммы.xlsx')

worksheet = workbook.add_worksheet()


# Данные

data = [10, 40, 50, 20, 10, 50, 20]

worksheet.write_column('A1', data)


# Тип диаграммы

chart = workbook.add_chart({'type': 'line'})


# Строим по нашим данным
```

```
chart.add_series({'values': '=Sheet1!A1:A7'})


worksheet.insert_chart('C1', chart)

workbook.close()
```



С помощью библиотеки openpyxl можно как создавать, так и читать и редактировать файлы формата xlsx. Рассмотрите примеры использования этой библиотеки самостоятельно.