Лекция 3. Программное заполнение документов Word и Excel

Для работы с документами Word будем использовать 2 основных пакета (библиотеки):

- ✓ docxtpl для управления переменными, вставленными в шаблон docx;
- ✓ python-docx для чтения, записи и создания документов (мощен для создания документов, но сложен для их изменения).

pip install docxtpl

Использование:

Сначала необходимо создать документ-шаблон и сохранить его в формате .docx (Microsoft Word 2007 и старше).

Документ должен быть полностью отформатирован. Там, где в нем должен быть другой текст (выполнена замена текста), следует вставить «переменные», которые заключаются в двойные фигурные скобки, не содержат пробелы или знаки препинания (например, {{имя_переменной}}). Для всех одинаковых переменных будет выполнена их замена одним и тем же текстом. Формат сохраняется прежний. Если для некоторых переменных не указать их замену, то они будут удалены.

Для замены создается словарь, в котором ключ – имя переменной, значение – замещающий текст.

После выполнения замены документ нужно сохранить. Лучше, если имя документа будет другое. Месторасположение документа также можно изменить (обычно все шаблоны хранятся в отдельной папке).

Пример 1.

```
from docxtpl import DocxTemplate
doc = DocxTemplate("my_word_template.docx")
context = { 'company_name' : "World company" }
doc.render(context)
doc.save("generated_doc.docx")
```

Работа с библиотекой python-docx

Установка пакета:

```
pip install python-docx
```

Подключение к проекту и создание нового документа:

```
from docx import Document
document = Document()
```

Откроет ***пустой документ***, основанный на «шаблоне» по умолчанию. Можно открывать и работать с существующим документом Word.

Параграфы (***абзацы***) являются основными объектами в Word. Они используются для основного текста, а также для заголовков и элементов списка.

Добавление абзаца:

```
add_paragraph(str text[, str style])

paragraph = document.add_paragraph('Lorem ipsum dolor sit amet.')
```

Этот метод возвращает ссылку на абзац, недавно добавленный *абзац в конце документа*. Для переменной paragraph этом случае назначается ссылка на этот абзац.

Также можно использовать один абзац в качестве «курсора» и ***вставить новый*** абзац прямо над ним:

```
prior_paragraph = paragraph.insert_paragraph_before('Lorem ipsum')
```

Это позволяет вставлять абзац в середину документа, что часто важно при изменении существующего документа, а не при его создании с нуля.

Добавление заголовка

В любом документе, кроме самого короткого, основной текст делится на разделы, каждый из которых начинается с заголовка. Так можно добавить заголовок:

```
document.add_heading('The REAL meaning of the universe')
```

По умолчанию это добавляет заголовок верхнего уровня, который отображается в Word как «Заголовок 1». Если нужны заголовки для подразделов, то следует указывать желаемый уровень целым числом от 1 до 9:

```
document.add_heading('The role of dolphins', level=2)
```

Если указать уровень 0, добавляется абзац «Заголовок». Это может быть удобно для создания относительно короткого документа, который не имеет отдельной титульной страницы.

Пример программы с использованием python-docx и документация к нему: https://python-docx.readthedocs.io/en/latest/

Форматирование отдельных слов в абзаце:

```
paragraph = document.add_paragraph('Lorem ipsum ')
run = paragraph.add_run('dolor')
run.bold = True
paragraph.add_run(' sit amet.')
document.save('test.docx')
```

Добавление таблицы

Часто встречается контент, который поддается табличному представлению, выстроенному в аккуратные строки и столбцы. Word довольно хорошо справляется с этой задачей. Так можно добавить таблицу:

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

Таблицы имеют несколько свойств и методов, которые используются для их заполнения. Доступ к ячейке по номерам строки и столбца:

```
cell = table.cell(0, 1)
```

Это позволит обратиться к верхней строке таблицы, которая должна быть предварительно создана. Обратите внимание, что указатели на строки и столбцы начинаются с нуля, как и при доступе к списку.

В ячейку можно поместить текст:

```
cell.text = 'parrot, possibly dead'
```

Зачастую проще получить доступ к ряду ячеек одновременно, например, при заполнении таблицы переменной длины из источника данных. Свойство таблицы –

rows, обеспечивает доступ к отдельным строкам, каждая из которых имеет свойство cells. Например:

```
row = table.rows[1]
row.cells[0].text = 'Foo bar to you.'
row.cells[1].text = 'And a hearty foo bar to you too sir!'
```

Коллекции rows и columns в таблице являются итеративными, поэтому их можно использовать непосредственно в for цикле. То же самое с cells последовательностями в строке или столбце:

```
for row in table.rows:
   for cell in row.cells:
      print(cell.text)
```

Выход за пределы допустимых значений индексов считается большой ошибкой. Поэтому прежде чем использовать номер строки и столбца ячейки, надо быть уверенным в его наличии. Количество строк или столбцов в таблице можно получить с помощью функции len():

```
row_count = len(table.rows)
col_count = len(table.columns)
```

Считывание всех абзацев из документа:

```
p = document.paragraphs
```

Считывание всех таблиц из документа:

```
t = document.tables
```

Запись данных в ячейку таблицы:

```
t[0].rows[1].cells[2].text = str(25)
table.cell(0,0).width = 1.2
```

Пример 2.

```
from docx import Document
document = Document('test.docx')
par = document.paragraphs
doc = Document()
for x in par:
    print(x.text)

tab = document.tables
for row in tab[0].rows:
    for cell in row.cells:
        print(cell.text, end = " ")
    print()
print('kol tables = ', len(tab))
n = len(tab[0].columns)
print('kol cells = ', n)
for i in range(3):
    row = tab[0].add_row()
    for j in range(n):
        row.cells[j].text = str(i) + str(j)

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

Добавление столбца таблицы справа:

```
add_column()
add_column(Inches(1.0))
```

Для копирования параграфа из одного документа в другой с сохранением форматирования, необходимо по факту взять кусок XML и вставить его в другом документе.

Порядок выполнения действий:

Создать новый параграф, в который будем копировать

```
# p - новый параграф куда копируем
p = new_doc.add_paragraph()
```

Берём текущий параграф и получаем доступ к атрибуту _element. После этого будем находиться внутри Параграфа, а именно внутри тэга в XML разметке

```
elem = old_p._element
```

Остаётся в цикле обойти все подтэги внутри старого параграфа и скопировать их в новый.

```
p_el=p._element
for i in elem.getchildren():
    p_el.append(i)
```

Пример 3.

```
document1 = Document()
for x in par:
    p = document1.add_paragraph()
    p_el=p._element
    elem = x._element
    for i in elem.getchildren():
        p_el.append(i)

document1.save('test1.docx')
```

Другие полезные библиотеки

Пакет можно использовать для перевода числовых значений денежных единиц в строковый тип:

```
from number_to_string import get_string_by_number
print(get string by number(3212.01))
```

Результат: Три тысячи двести двенадцать рублей 01 копейка

Пакет datetime содержит следующие классы:
datetime.date, datetime.timedelta, datetime.datetime

Также существует класс tzinfo, который применяется для работы с часовыми поясами.

datetime.date

Python может представлять даты различными способами. Для начала, рассмотрим формат datetime.date, так как это один из самых простых объектов date.

```
import datetime
print(datetime.date.today()) # 2020-02-13
```

Пример 4.

```
import datetime

a = datetime.datetime(2020, 3, 5)
print(a)  # 2020-03-05 00:00:00

b = datetime.datetime(2020, 3, 5, 12, 30, 10)
print(b)  # 2017-03-05 12:30:10

d = datetime.datetime(2020, 3, 5, 12, 30, 10)
print(d.year)  # 2020
print("%02d" % d.month)  # 03
print("%02d" % d.day)  # 05
print("%02d" % d.second)  # 10
print("%02d" % d.hour)  # 12
print(d.second)
```

Переменной **d** присваиваем объект **datetime**. Теперь можно получить доступ к различным компонентам даты по названиям, таким как **d.year** или **d.month**.

datetime.datetime принимает несколько дополнительных аргументов: *год, месяц, день, час, минута и секунда*. Он также позволяет указывать информацию о **микросекундах** и **часовом поясе**.

today совместно с datetime.datetime использует два разных метода:

```
a = datetime.datetime.today()
print(a)  # 2020-02-13 01:39:00.842111
print(a.day)  # 13
print(a.month)  # 2

b = datetime.datetime.now()
print(b)  # результат такой же

a = datetime.datetime.today().strftime("%d.%m.%Y")
print(a)  # 13.02.2020

today = datetime.datetime.today()
print(today.strftime("%m/%d/%Y"))  # 02/13/2020
```

Работа с табличными документами

Openpyxl – это библиотека Python для чтения и записи файлов Excel (с расширением xlsx / xlsm / xltx / xltm). Модуль openpyxl позволяет программе Python читать и изменять файлы Excel.

Например, пользователю, возможно, придется пройти через тысячи строк и выбрать несколько полезных сведений, чтобы внести небольшие изменения на основе некоторых критериев. Используя модуль **openpyxl**, эти задачи могут быть выполнены очень эффективно и легко.

```
import openpyxl

# Вызов функции Workbook () из openpyxl
# создать новый пустой объект Workbook
wb = openpyxl.Workbook()
```

Описание некоторых действий при работе с электронными таблицами

Получить рабочий лист из активного атрибута.

```
wb = openpyxl.Workbook()
sheet = wb.active
sheet.title = "Sheet1"
sheet["A1"].value = "Yes"
wb.create_sheet(index=0, title="Sheet2")
wb.save("test1.xlsx")
```

Установка размеров ячеек:

```
# установить высоту строки
sheet.row_dimensions[1].height = 70

# установить ширину столбца
sheet.column_dimensions['B'].width = 20
```

Объединение ячеек:

```
# A2: D4 'объединяет 12 ячеек в одну ячейку.
sheet.merge_cells('A2:D4')

sheet.cell(row=2, column=1).value = 'Twelve cells join together.'
# снять объединение ячеек
sheet.unmerge_cells('A2:D4')

from openpyxl.styles import Font

sheet.cell(row=1, column=1).value = "Ankit Rai"

# установить размер ячейки 24
sheet.cell(row=1, column=1).font = Font(size=24)

sheet.cell(row=2, column=2).value = "Ankit Rai"

# установить стиль шрифта курсивом
sheet.cell(row=2, column=2).font = Font(size=24, italic=True)

sheet.cell(row=3, column=3).value = "Ankit Rai"

# установить стиль шрифта жирным шрифтом
sheet.cell(row=3, column=3).font = Font(size=24, bold=True)

sheet.cell(row=4, column=4).value = "Ankit Rai"

# установить имя шрифта 'Times New Roman'
sheet.cell(row=4, column=4).font = Font(size=24, name='Times New
Roman')
```

Пример 5.

```
import openpyxl
tab1 = openpyxl.load_workbook("test2.xlsx")
sheets = tab1.sheetnames  # get_sheet_names()
sheet1 = tab1["data"]  # tab1.active
# excel_tabs = openpyxl.workbook("test2.xlsx")
# tab1.create_sheet(index=1, title="demo sheet2")
```

```
print(sheets[0])
print(sheet1.title)
print(sheet1["A1"].value)  # 1
print(sheet1["A1"].column)  # 1
print(sheet1["A1"].row)
# 1 если ввести sheets[0]["A1"].row, то ошибка
print(sheet1["B1"].coordinate)  # B1
print(sheet1.cell(row=1, column=2))  # <Cell 'data'.B1>
print(sheet1.cell(row=1, column=2).value)  # 2
# for i in range(1, 15):
#     print(sheet1.cell(row=i, column=1).value)
# если ячейка пустая, то ее значение None

for i in range(1, sheet1.max_row + 1):
    for j in range(1, sheet1.max_column + 1):
        cell = sheet1.cell(row=i, column=j).value
        if cell != None:
            print(cell, end=" ")
        else:
            print("*", end=" ")
    print()

for line in sheet1["A1": "C10"]:
    for cell in line:
        print(cell.value, end=" ")
    print()

dict1 = {}
sheet2 = tab1["Лист2"]
x = sheet2.cell(row=sheet2.max_row,
column=sheet2.max_column).coordinate
print(x)
for line in sheet2["A1": x]:
    dict1[line[0].value] = line[1].value
    # for cell in line:
    #     print(cell.value, end=" ")
    # print()
print(dict1)
x = sheet1.cell(row=sheet1.max_row,
column=sheet1.max_column).coordinate
for line in sheet1["a1": x]:
    for cell in line:
        if cell.value == None:
            cell.value = "=PRODUCT(A1:A5)"
            #"= SUM(A1:A5)""=A2+B1"'= AVERAGE(A1:A5)'

sheet2.append(["06", "Отдел маркетинга"])
print(sheet2["c1"].value)
if isinstance(sheet2["c1"].value, int):
    print("int")
elif isinstance(sheet2["c1"].value, str):
    print("str")
else:
    print("*")
tab1.save("test2.xlsx")
```