Материалы занятия

Курс: Разработка Web-приложений на Python, с применением Фреймворка Django
Дисциплина: Основы программирования на Python

Тема занятия №11: Работа с комплексными файлами - excel, json, word. Git и Jira

1. Работа с библиотекой openpyxl – открытие, чтение, запись и форматирование файлов

Электронные таблицы Excel — это интуитивно понятный и удобный способ манипулирования большими наборами данных без какой-либо предварительной технической подготовки. По этому, это один из форматов, с которым, в какой-то момент времени, вам придется иметь дело. Часто будут стоять задачи по извлечению каких-то данных из базы данных или файла логов в электронную таблицу Excel, или наоборот, преобразовывать электронную таблицу Excel в какую-либо более удобную программную форму, примеров этому масса.

Модуль openpyxl — это библиотека Python для чтения/записи форматов Office Open XML (файлов Excel 2010) с расширениями xlsx/xlsm/xltx/xltm.

Установка модуля openpyxl в виртуальное окружение.
Модуль openpyxl 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 openpyxl
(VirtualEnv):~$ python3 -m pip install -U openpyxl
```

Создание книги Excel.
Чтобы начать работу с модулем openpyxl, нет необходимости создавать файл электронной таблицы в файловой системе. Нужно просто импортировать класс Workbook и создать его экземпляр. Рабочая книга всегда создается как минимум с одним рабочим листом, его можно получить, используя свойство Workbook.active:

```
>>> from openpyxl import Workbook
# создаем книгу
>>> wb = Workbook()
# делаем единственный лист активным
>>> ws = wb.active
```

Новый рабочий лист книги Excel.
Новые рабочие листы можно создавать, используя метод Workbook.create_sheet():

```
# вставить рабочий лист в конец (по умолчанию)
>>> ws1 = wb.create_sheet("Mysheet")
# вставить рабочий лист в первую позицию
>>> ws2 = wb.create_sheet("Mysheet", 0)
# вставить рабочий лист в предпоследнюю позицию
>>> ws3 = wb.create_sheet("Mysheet", -1)
```

Доступ к ячейке и ее значению.
После того как выбран рабочий лист, можно начинать изменять содержимое ячеек. К ячейкам можно обращаться непосредственно как к ключам рабочего листа, например ws['A4']. Это вернет ячейку на A4 или создаст ее, если она еще не существует. Значения могут быть присвоены напрямую:

```
>>> ws['A4'] = 5
>>> ws['A4']
# <Cell 'NewPage'.A4>
>>> ws['A4'].value
# 5
>>> ws['A4'].column
# 1
>>> ws['A4'].row
# 4
```

Если объект ячейки присвоить переменной, то этой переменной, также можно присваивать значение:

```
>>> c = ws['A4']
>>> c.value = c.value * 2
>>> c.value
# 10
```

Доступ к диапазону ячеек листа электронной таблицы.
Диапазон с ячейками активного листа электронной таблицы можно получить с помощью простых срезов. Эти срезы будут возвращать итераторы объектов ячеек.

```
>>> cell_range = ws['A1':'C2']
>>> cell_range
# ((<Cell 'NewPage'.A1>, <Cell 'NewPage'.B1>, <Cell 'NewPage'.C1>),
# (<Cell 'NewPage'.A2>, <Cell 'NewPage'.B2>, <Cell 'NewPage'.C2>))
```

Сохранение созданной книги в файл Excel.
Самый простой и безопасный способ сохранить книгу, это использовать метод Workbook.save() объекта Workbook:

```
>>> wb = Workbook()
>>> wb.save('test.xlsx')
```

Внимание. Эта операция перезапишет существующий файл без предупреждения!!!
После сохранения, можно открыть полученный файл в Excel и посмотреть данные, выбрав лист с именем NewPage.
Примечание. Расширение имени файла не обязательно должно быть xlsx или xlsm, хотя могут возникнуть проблемы с его открытием непосредственно в другом приложении. Поскольку файлы OOXML в основном представляют собой ZIP-файлы, их также можете открыть с помощью своего любимого менеджера ZIP-архивов.

Загрузка документа XLSX из файла.
Чтобы открыть существующую книгу Excel необходимо использовать функцию openpyxl.load_workbook():

```
>>> from openpyxl import load_workbook
>>> wb2 = load_workbook('test.xlsx')
>>> print(wb2.sheetnames)
# ['Mysheet1', 'NewPage', 'Mysheet2', 'Mysheet']
```

Ещё пример для работы с excel:

```
new_workbook = openpyxl.Workbook()
new_worksheet = new_workbook.active
new_worksheet.title = 'page 1'
rows = [[f"{column_index} {row_index}" for column_index in "ABCDEFGH"] for row_index in range(1, 10)]
index_row1 = 0
for row_index, row in enumerate(rows, 1):
    for column_index, value in enumerate(row, 1):
        # worksheet[f"{get_column_letter(column_index)}{row_index}"] = value
        new_worksheet.cell(row_index, column_index, value)
new_workbook.save("temp/new_data.xlsx")

# a1 = worksheet[f"A1"]
# a1.font = openpyxl.styles.Font(color="00FF6600", bold=True, sz=28)
# worksheet[f"A1"] = "Фамилия Имя"
# worksheet.merge_cells('B2:C6')
# top_left_cell = worksheet['B2']
# top_left_cell.value = "My Cell"
# thin = openpyxl.styles.Side(border_style="thin", color="000000")
# double = openpyxl.styles.Side(border_style="double", color="ff0000")
# top_left_cell.border = openpyxl.styles.Border(top=double, left=thin, right=thin, bottom=double)
# top_left_cell.fill = openpyxl.styles.PatternFill("solid", fgColor="DDDDDD")
# top_left_cell.fill = fill = openpyxl.styles.GradientFill(stop=("000000", "FFFFFF"))
# top_left_cell.font = openpyxl.styles.Font(b=True, color="FF0000")
# top_left_cell.alignment = openpyxl.styles.Alignment(horizontal="center", vertical="center")

# font = openpyxl.styles.Font(name='Tahoma', size=16, bold=True,
#                             italic=False, vertAlign=None, underline='none',
#                             strike=False, color='FF0000FF')
# worksheet['B2'].font = font
# worksheet['B2'] = 'Python'
```

2. Работа с библиотекой json – открытие, чтение и запись

Python поддерживает JSON

Python содержит встроенный модуль под названием json для кодирования и декодирования данных JSON.

```
1 import json
```

Небольшой словарь

Как правило, процесс кодирования JSON называется сериализация. Этот термин обозначает трансформацию данных в серию байтов (следовательно, серийных) для хранения или передачи по сети. Также вы, возможно, уже слышали о термине «маршалинг», но это уже совсем другая область.

Естественно, десериализация — является противоположным процессом декодирования данных, которые хранятся или направлены в стандарт JSON.

Сериализация JSON

Что происходит после того, как компьютер обрабатывает большие объемы информации? Ему нужно принять дамп данных. Соответственно, модуль json предоставляет метод dump() для записи данных в файлы. Также есть метод dumps() для записей в строку Python.

Простые объекты Python переводятся в JSON согласно с весьма интуитивной конверсией.

Python	JSON
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	null

Пример сериализации JSON Python

data_file.json

```
1 data = {
2     "president": {
3         "name": "Zaphod Beeblebrox",
4         "species": "Betelgeusian"
5     }
6 }
```

Сохранить эту информацию на диск — критично, так что ваша задача — записать на файл.

Используя контекстный менеджер Python, вы можете создать файл под названием data_file.json и открыть его в режиме write (файлы JSON имеют расширение .json).

```
with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

Обратите внимание на то, что dump() принимает два позиционных аргумента: (1) объект данных, который сериализуется и (2), файловый объект, в который будут вписаны байты.

Или, если вы склонны продолжать использовать эти сериализированные данные JSON в вашей программе, вы можете работать как со строкой.

```
json_string = json.dumps(data)
```

Обратите внимание, что файловый объект является пустым, так как вы на самом деле не выполняете запись на диск. Кроме того, dumps() аналогичен dump().

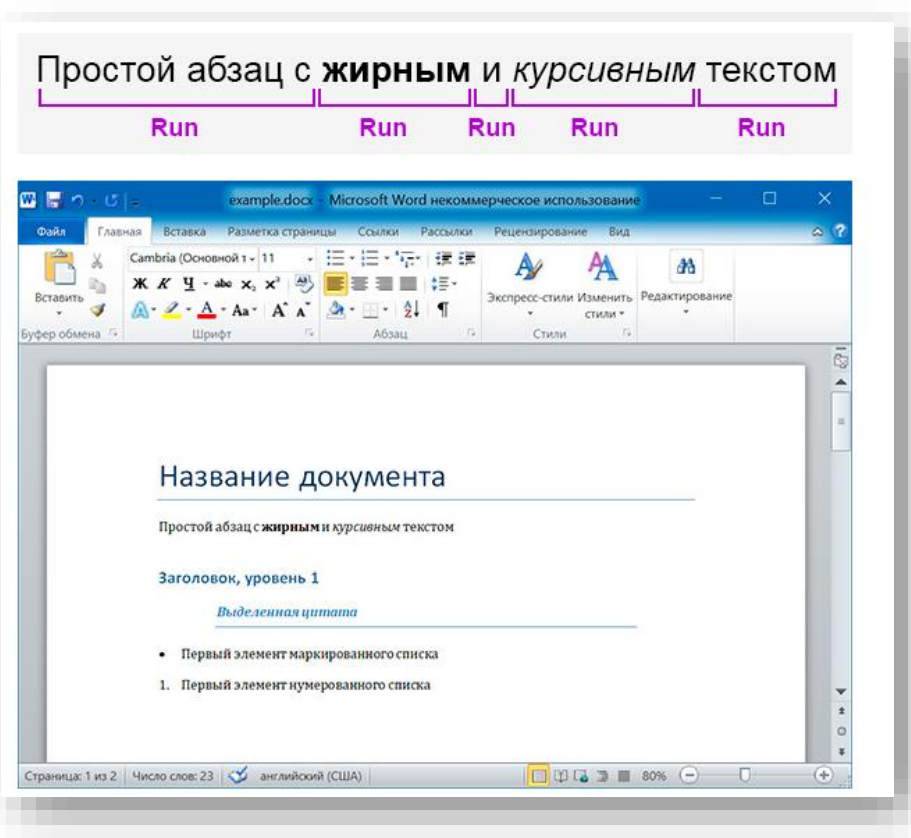
1. Работа с библиотекой word – открытие, чтение и запись

С помощью модуля python-docx можно создавать и изменять документы MS Word с расширением .docx. Чтобы установить этот модуль, выполняем команду

```
> pip install python-docx
```

Чтение документов MS Word

Файлы с расширением .docx обладают развитой внутренней структурой. В модуле python-docx эта структура 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.




```python
import docx

doc = docx.Document('example.docx')

# количество абзацев в документе
print(len(doc.paragraphs))

# текст первого абзаца в документе
print(doc.paragraphs[0].text)

# текст второго абзаца в документе
print(doc.paragraphs[1].text)

# текст первого Run второго абзаца
print(doc.paragraphs[1].runs[0].text)
```

```
6
Название документа
Простой абзац с жирным и курсивным текстом
Простой абзац с
```

Получаем весь текст из документа:

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

```
Название документа
Простой абзац с жирным и курсивным текстом
Заголовок, уровень 1
Выделенная цитата
Первый элемент маркированного списка
Первый элемент нумерованного списка
```

Атрибуты объекта Run

Отдельные фрагменты текста, представленные объектами Run, могут подвергаться дополнительному форматированию с помощью атрибутов. Для каждого из этих атрибутов может быть задано одно из трех значений: True (атрибут активизирован), False (атрибут отключен) и None (применяется стиль, установленный для данного объекта Run).

- `bold` — Полужирное начертание
- `underline` — Подчеркнутый текст
- `italic` — Курсивное начертание
- `strike` — Зачеркнутый текст

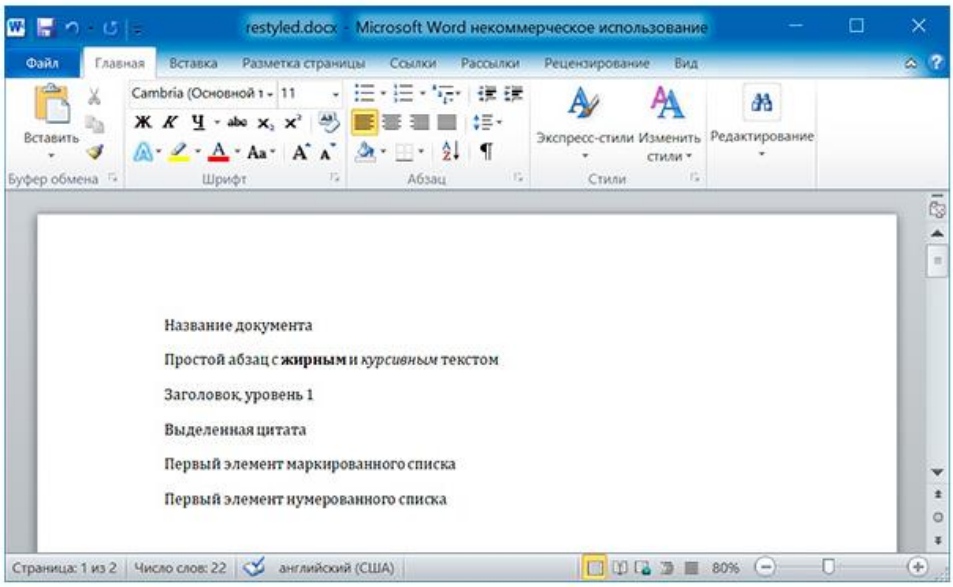
Изменим стили для всех параграфов нашего документа:

```python
import docx

doc = docx.Document('example.docx')

# изменяем стили для всех параграфов
for paragraph in doc.paragraphs:
    paragraph.style = 'Normal'

doc.save('restyled.docx')
```



Запись документов MS Word

Добавление абзацев осуществляется вызовом метода add_paragraph() объекта Document. Для добавления текста в конец существующего абзаца, надо вызвать метод add_run() объекта Paragraph:

```python
import docx

doc = docx.Document()

# добавляем первый параграф
doc.add_paragraph('Здравствуй, мир!')

# добавляем еще два параграфа
par1 = doc.add_paragraph('Это второй абзац.')
par2 = doc.add_paragraph('Это третий абзац.')

# добавляем текст во второй параграф
par1.add_run(' Этот текст был добавлен во второй абзац.')

# добавляем текст в третий параграф
par2.add_run(' Добавляем текст в третий абзац.').bold = True

doc.save('helloworld.docx')
```