Python Programing

Lesson 06: Work with Word, Excel and XML Files

People matter, results count.

Lesson Objectives

- After completing this lesson, you will learn about:
  - How to process Excel Files?
  - How to process Word Files?
  - How to process XML Files?

Processing Excel Files

- The openpyxl module allows your Python programs to read and modify Excel spreadsheet files.
  - For example:
  - A) You might have the boring task of copying certain data from one spreadsheet and pasting it into another one.
  - B) You might have to go through thousands of rows and pick out just a handful of them to make small edits based on some criteria.
  - C) You might have to look through hundreds of spreadsheets of department budgets, searching for any that are in the red.

➢ **Python can automate all such task with the help of openpyxl module.**

Basic Definitions

- An Excel spreadsheet document is called a workbook. A single workbook is saved in a file with the .xlsx extension.
- Each workbook can contain multiple sheets(also called worksheets). The sheet the user is currently viewing (or last viewed before closing Excel) is called the active sheet.
- Each sheet has columns (addressed by letters starting at A) and rows (addressed by numbers starting at 1).
- A box at a particular column and row is called a cell. Each cell can contain a number or text value. The grid of cells with data makes up a sheet.

Installing the openpyxl Module

- Python does not come with OpenPyXL, so you’ll have to install it.
- Openpyxl module is dependent on jdcal and et_xmlfile modules.
- Add pathnames of all modules mentioned to sys in the following order.

```
>> import sys
>> sys.path.append('d:\\python\\jdcal-1.2')
>> sys.path.append('d:\\python\\et_xmlfile-1.0.1')
>> sys.path.append('d:\\python\\openpyxl-2.3.3')
```

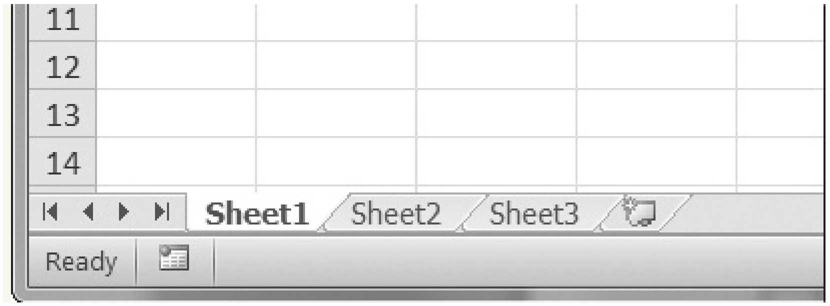
Execute following statement:

```
>> import openpyxl
```

- OR
  - Execute ‘pip install openpyxl’ command

Example

- Below figure shows the tabs for the three default sheets named Sheet1, Sheet2, and Sheet3 that Excel automatically provides for new workbooks



- Enter following data in Sheet 1 and save the excel file as Example.xlsx

	A	B	C
1	4/5/2015 13:34	Apples	73
2	4/5/2015 3:41	Cherries	85
3	4/6/2015 12:46	Pears	14
4	4/8/2015 8:59	Oranges	52
5	4/10/2015 2:07	Apples	152
6	4/10/2015 18:10	Bananas	23
7	4/10/2015 2:40	Strawberries	98

Opening Excel Document using OpenPyXL

- Once you have imported openpyxl module, you can now use openpyxl.load_workbook() function.
- Execute following on Python interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

- The openpyxl.load_workbook() function takes in the filename and returns a value of the workbook data type. This Workbook object represents the Excel file, a bit like how a File object represents an opened text file.

Getting Sheets from Workbook

- You can get a list of all the sheet names in the workbook by calling the get_sheet_names() method.
- Each sheet is represented by a Worksheet object, which you can obtain by passing the sheet name string to the get_sheet_by_name() workbook method.
- Finally, you can read the active member variable of a Workbook object to get the workbook’s active sheet.
  - The active sheet is the sheet that’s on top when the workbook is opened in Excel. Once you have the Worksheet object, you can get its name from the title attribute.

Getting Cells from Sheet

- Once you have a Worksheet object, you can access a Cell object by its name.
- The Cell object has a value attribute that contains, unsurprisingly, the value stored in that cell.
- Cellobjects also have row, column, and coordinate attributes that provide location information for the cell.

Converting Between Column Letters and Numbers

- To convert from letters to numbers, call the openpyxl.cell.column_index_from_string()function.

- To convert from numbers to letters, call the openpyxl.cell.get_column_letter()function.

Getting Rows and Columns from the Sheets

- You can slice Worksheet objects to get all the Cell objects in a row, column, or rectangular area of the spreadsheet. Then you can loop over all the cells in the slice.

```
>>> import openpyxl
   >>> wb = openpyxl.load_workbook('example.xlsx')
   >>> sheet = wb.get_sheet_by_name('Sheet1')
   >>> tuple(sheet['A1':'C3'])
   ((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell
Sheet1.A2>,
   <Cell Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell
Sheet1.B3>,
   <Cell Sheet1.C3>))
   >>> for rowOfCellObjects in sheet['A1':'C3']:
             for cellObj in rowOfCellObjects:
               print(cellObj.coordinate, cellObj.value)
           print('--- END OF ROW ---')
```

WorkBooks, Sheets and Cells - Steps

- Follow below mentioned steps:

1. Import the openpyxl module.
2. Call the openpyxl.load_workbook() function.
3. Get a Workbook object.
4. Read the active member variable or call the get_sheet_by_name() workbook method.
5. Get a Worksheet object.
6. Use indexing or the cell() sheet method with row and column keyword arguments.
7. Get a Cell object.
8. Read the Cell object's value attribute.

Writing Excel Documents

- OpenPyXL also provides ways of writing data, meaning that your programs can create and edit spreadsheet files. With Python, it’s simple to create spreadsheets with thousands of rows of data.
- Creating and Saving Excel Documents
  - Call the openpyxl.Workbook() function to create a new, blank Workbook object. Enter the following into the interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> sheet = wb.active
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'
>>> wb.get_sheet_names()
['Spam Bacon Eggs Sheet']
```

Writing Excel Documents

- Creating and Removing Sheets
  - Sheets can be added to and removed from a workbook with the create_sheet() and remove_sheet() methods.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.get_sheet_names()
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='First Sheet')
<Worksheet "First Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Middle Sheet')
<Worksheet "Middle Sheet">
>>> wb.get_sheet_names()
['First Sheet', 'Sheet', 'Middle Sheet', 'Sheet1']
```

Writing Excel Documents

- Writing values to cells
  - Writing values to cells is much like writing values to keys in a dictionary. Enter this into the interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> sheet['A1'] = 'Hello world!'
>>> sheet['A1'].value
```

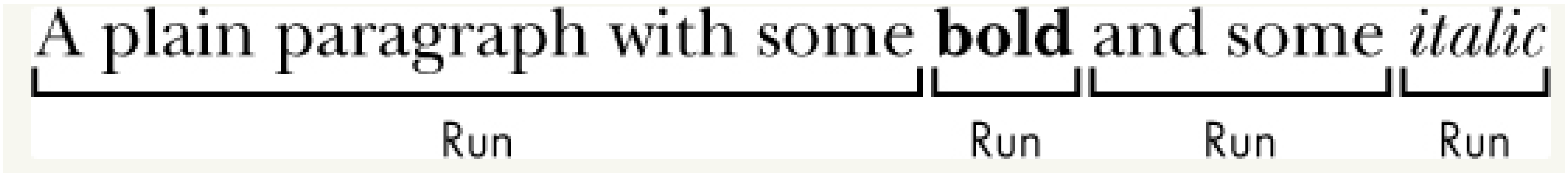
- 'Hello world!'

Processing Word Files

- Python can create and modify Word documents, which have the .docx file extension, with the python-docx module.
- You can install the module by running pip install python-docx.
- The full documentation for Python-Docx is available at https://python-docx.readthedocs.org/

Processing Word Files

- Compared to plaintext, .docx files have a lot of structure.
  - This structure is represented by three different data types in Python-Docx.
  - At the highest level, a Document object represents the entire document. The Document object contains a list of Paragraph objects for the paragraphs in the document. (A new paragraph begins whenever the user presses ENTER or RETURN while typing in a Word document.)
  - Each of these Paragraph objects contains a list of one or more Run objects.



Demo

- Reading Word Documents

```
>>> import docx
❶ >>> doc = docx.Document('demo.docx')
❷ >>> len(doc.paragraphs)
   7
❸ >>> doc.paragraphs[0].text
   'Document Title'
❹ >>> doc.paragraphs[1].text
   'A plain paragraph with some bold and
some italic'
❺ >>> len(doc.paragraphs[1].runs)
   4
❻ >>> doc.paragraphs[1].runs[0].text
   'A plain paragraph with some '
❼ >>> doc.paragraphs[1].runs[1].text
   'bold'
❽ >>> doc.paragraphs[1].runs[2].text
   ' and some '
❾ >>> doc.paragraphs[1].runs[3].text
   'italic'
```

Getting Full text from a .docx file

- If you care only about the text, not the styling information, in the Word document, you can use the getText() function.
- It accepts a filename of a .docx file and returns a single string value of its text.

```
def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

```
>>> import readDocx
>>>
print(readDocx.getText('demo.docx
'))
```

Demo

- Getting Full text from a .docx file

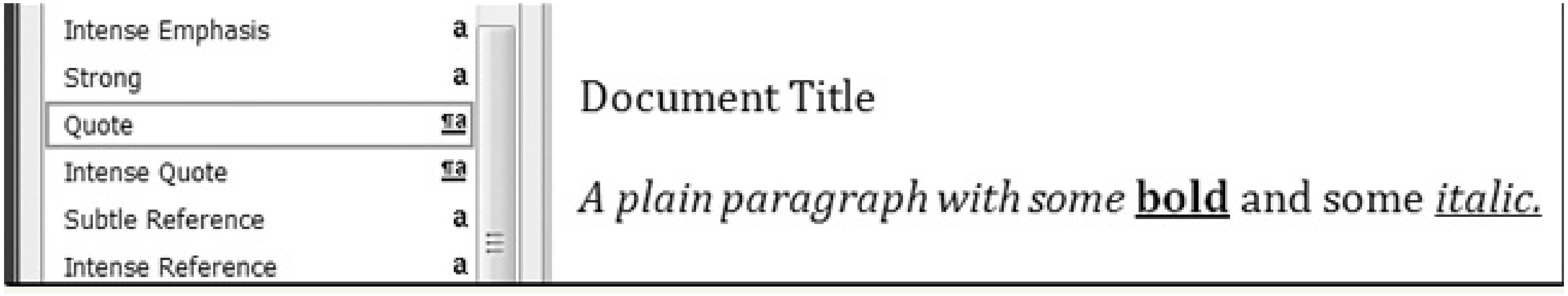
Run Attributes

- Runs can be further styled using text attributes.
- Each attribute can be set to one of three values:
  - True(the attribute is always enabled, no matter what other styles are applied to the run).
  - False (the attribute is always disabled).
  - None (defaults to whatever the run's style is set to).

Attribute	Description
bold	The text appears in bold.
italic	The text appears in italic.
underline	The text is underlined.
strike	The text appears with strikethrough.
double_strike	The text appears with double strikethrough.
all_caps	The text appears in capital letters.
small_caps	The text appears in capital letters, with lowercase letters two points smaller.
shadow	The text appears with a shadow.
outline	The text appears outlined rather than solid.
rtl	The text is written right-to-left.
imprint	The text appears pressed into the page.
emboss	The text appears raised off the page in relief.

Run Attributes

- Here we show how the styles of paragraphs and runs look in restyled.docx.



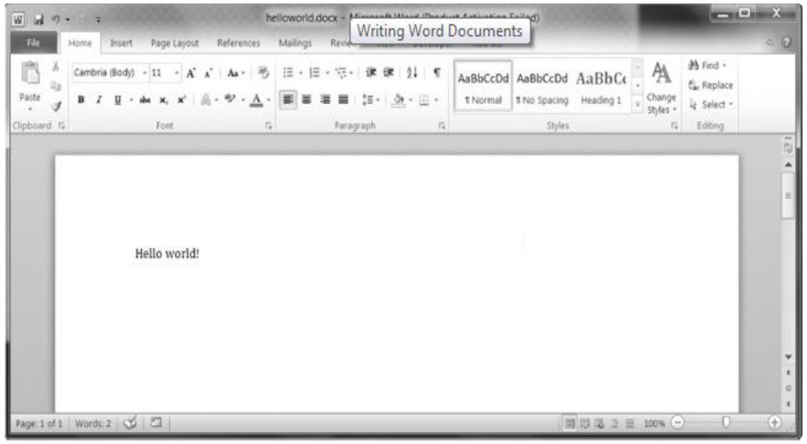
Document Title

A plain paragraph with some **bold** and some *italic.*

Writing Word Documents

- This code will create a file named helloworld.docx in the current working directory that, when opened, looks as follows:

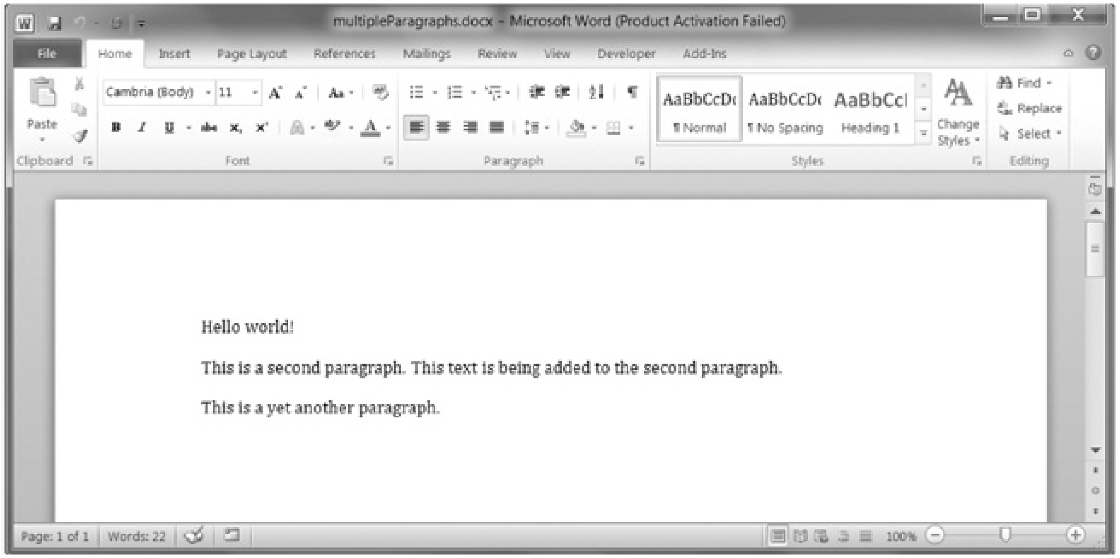
```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
>>> doc.save('helloworld.docx')
```



Hello world!

Writing Word Documents

- You can add paragraphs by calling the add_paragraph() method again with the new paragraph's text. Or to add text to the end of an existing paragraph, you can call the paragraph's add_run() method and pass it a string.
- Enter the code shown in Notes page into the interactive shell:
- Resulting document looks as follows:



Hello world!

This is a second paragraph. This text is being added to the second paragraph.

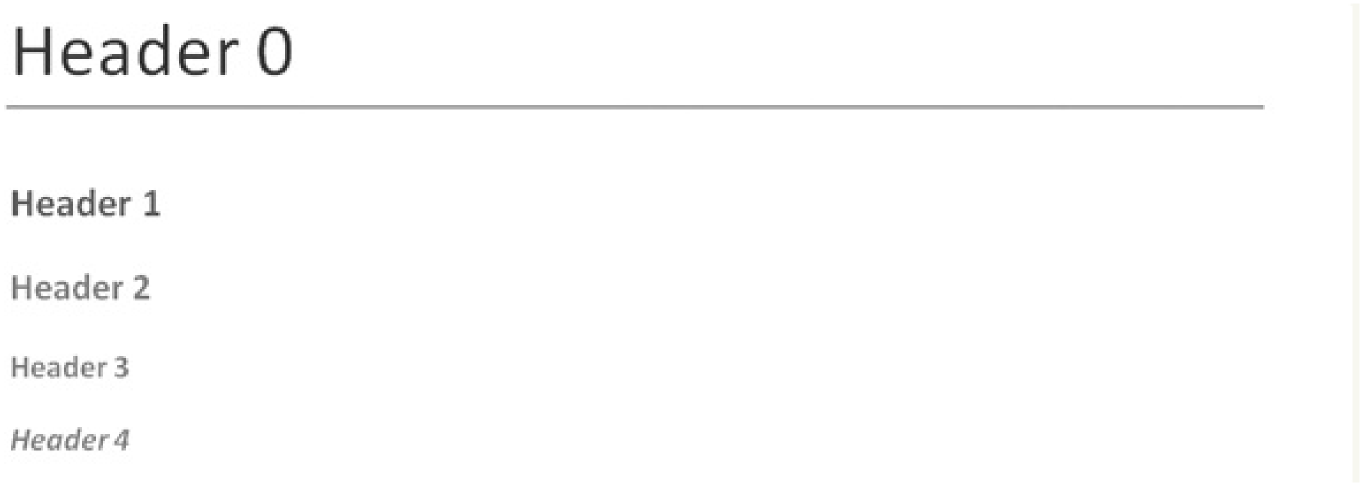
This is a yet another paragraph.

Writing Word Documents

- You can add paragraphs by calling the add_paragraph() method again with the new paragraph’s text. Or to add text to the end of an existing paragraph, you can call the paragraph’s add_run() method and pass it a string.
- Enter the code shown in Notes page into the interactive shell:

Writing Word Documents – Adding Headings

- Calling add_heading() adds a paragraph with one of the heading styles. Enter the code in Notes page into the interactive shell.
- The resulting headings.docx file will looks as follows:



Header 0

Header 1

Header 2

Header 3

Header 4

Writing Word Documents – Adding line and Page Breaks

- To add a line break (rather than starting a whole new paragraph), you can call the add_break()method on the Run object you want to have the break appear after. If you want to add a page break instead, you need to pass the value docx.text.WD_BREAK.PAGE as a lone argument toadd_break(), as is done in the middle of the following example:

```
>>> doc = docx.Document()
>>> doc.add_paragraph('This is on the first page!')
<docx.text.Paragraph object at 0x0000000003785518>
➊ >>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)
>>> doc.add_paragraph('This is on the second page!')<docx.text.Paragraph
object at 0x00000000037855F8>     >>> doc.save('twoPage.docx')
```

Writing Word Documents – Adding Pictures

- Document objects have an add_picture() method that will let you add an image to the end of the document. Say you have a file image.png in the current working directory. You can add image.png to the end of your document with a width of 1 inch and height of 4 centimeters (Word can use both imperial and metric units) by entering the following:

```
>>> doc.add_picture(‘image.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

Python XML Processing

- The Extensible Markup Language (XML) is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and developmental language.
- Python Standard library useful set of interfaces to work with XML.
- The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

Parsing XML with SAX APIs

- SAX is a standard interface for event-driven XML parsing. Parsing XML with SAX generally requires you to create your own ContentHandler by subclassing xml.sax.ContentHandler.
- Your ContentHandler handles the particular tags and attributes of your flavor(s) of XML. A ContentHandler object provides methods to handle various parsing events. Its owning parser calls ContentHandler methods as it parses the XML file.
- The methods startDocument and endDocument are called at the start and the end of the XML file. The method characters(text) is passed character data of the XML file via the parameter text.
- The ContentHandler is called at the start and end of each element. If the parser is not in namespace mode, the methods startElement(tag, attributes) andendElement(tag) are called; otherwise, the corresponding methods startElementNS and endElementNS are called. Here, tag is the element tag, and attributes is an Attributes object.

Important Methods

- The make_parser Method
  - It creates a new parser object and returns it. The parser object created will be of the first parser type the system finds.
    - xml.sax.make_parser([parser_list])
    - parser_list: The optional argument consisting of a list of parsers to use which must all implement the make_parser method.

Important Methods

- The parse Method
  - It creates a SAX parser and uses it to parse a document.
    - xml.sax.parse(xmlfile, contenthandler[, errorhandler])
  - Here is the detail of the parameters −
    - xmlfile: This is the name of the XML file to read from.
    - contenthandler: This must be a ContentHandler object.
    - errorhandler: If specified, errorhandler must be a SAX ErrorHandler object.

Important Methods

- The parseString Method
  - There is one more method to create a SAX parser and to parse the specified XML string.
  - xml.sax.parseString(xmlstring, contenthandler[, errorhandler])
  - Here is the detail of the parameters −
    - xmlstring: This is the name of the XML string to read from.
    - contenthandler: This must be a ContentHandler object.
    - errorhandler: If specified, errorhandler must be a SAX ErrorHandler object.

Demo

- Parsing XML with SAX.

Parsing XML with DOM APIs

- The Document Object Model ("DOM") is a cross-language API from the World Wide Web Consortium (W3C) for accessing and modifying XML documents.
- The DOM is extremely useful for random-access applications. SAX only allows you a view of one bit of the document at a time. If you are looking at one SAX element, you have no access to another.

Parsing XML with DOM APIs

```
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
movies = collection.getElementsByTagName("movie")
for movie in movies:
    if movie.hasAttribute("title")
        print "Title: %s" % movie.getAttribute("title")
```

- It shows easiest way to quickly load an XML document and to create a minidom object using the xml.dom module. The minidom object provides a simple parser method that quickly creates a DOM tree from the XML file.
- The parse(file [,parser]) function of the minidom object to parse the XML file designated by file into a DOM tree object.

Demo

- Processing xml using DOM parser

Summary

- In this lesson, you learnt:
  - How to process Excel Files
  - How to process Word Files
  - How to process XML Files