Manipulating a workbook in memory

Create a workbook

There is no need to create a file on the filesystem to get started with openpyxl. Just import the Workbook class and start using it

```
>>> from openpyxl import Workbook
>>> wb = Workbook()
```

A workbook is always created with at least one worksheet. You can get it by using the `openpyxl.workbook.Workbook.active()` property

```
>>> ws = wb.active
```

Note

This function uses the *_active_sheet_index* property, set to 0 by default. Unless you modify its value, you will always get the first worksheet by using this method.

You can also create new worksheets by using the `openpyxl.workbook.Workbook.create_sheet()` method

```
>>> ws1 = wb.create_sheet("Mysheet") # insert at the end (default)
# or
>>> ws2 = wb.create_sheet("Mysheet", 0) # insert at first position
```

Sheets are given a name automatically when they are created. They are numbered in sequence (Sheet, Sheet1, Sheet2, ...). You can change this name at any time with the *title* property:

```
ws.title = "New Title"
```

The background color of the tab holding this title is white by default. You can change this providing an RRGGBB color code to the sheet_properties.tabColor property:

```
ws.sheet_properties.tabColor = "1072BA"
```

Once you gave a worksheet a name, you can get it as a key of the workbook:

```
>>> ws3 = wb["New Title"]
```

You can review the names of all worksheets of the workbook with the `openpyxl.workbook.Workbook.sheetnames()` property

```
>>> print(wb.sheetnames)
['Sheet2', 'New Title', 'Sheet1']
```

？列表？

You can loop through worksheets

```
>>> for sheet in wb:
...     print(sheet.title)
```

You can create copies of worksheets within a single workbook:

`openpyxl.workbook.Workbook.copy_worksheet()` method:

```
>>> source = wb.active
>>> target = wb.copy_worksheet(source)
```

Note

Only cells and styles can be copied. You cannot copy worksheets between workbooks.

You can copy worksheets in a workbook with the

Playing with data

Accessing one cell

Now we know how to access a worksheet, we can start modifying cells content.

Cells can be accessed directly as keys of the worksheet

```
>>> c = ws['A4']
```

This will return the cell at A4 or create one if it does not exist yet. Values can be directly assigned

```
>>> ws['A4'] = 4
```

There is also the `openpyxl.worksheet.Worksheet.cell()` method.

This provides access to cells using row and column notation:

```
>>> d = ws.cell(row=4, column=2, value=10)
```

这种获取单元格的方法更适合于有变量的时候

Note

When a worksheet is created in memory, it contains no *cells*. They are created when first accessed.

Warning

Because of this feature, scrolling through cells instead of accessing them directly will create them all in memory, even if you don’t assign them a value.

Something like

```
>>> for i in range(1,101):
...        for j in range(1,101):
...            ws.cell(row=i, column=j)
```

will create 100x100 cells in memory, for nothing.

Accessing many cells

Ranges of cells can be accessed using slicing

```
>>> cell_range = ws['A1':'C2']
```

单元格区域可赋值给一个变量

Ranges of rows or columns can be obtained similarly:

```
>>> colC = ws['C']
>>> col_range = ws['C:D']
>>> row10 = ws[10]
>>> row_range = ws[5:10]
```

You can also use the `openpyxl.worksheet.Worksheet.iter_rows()` method:

```
>>> for row in ws.iter_rows(min_row=1, max_col=3, max_row=2):
...    for cell in row:
...        print(cell)
<Cell Sheet1.A1>
<Cell Sheet1.B1>
<Cell Sheet1.C1>
<Cell Sheet1.A2>
<Cell Sheet1.B2>
<Cell Sheet1.C2>
```

Likewise the `openpyxl.worksheet.Worksheet.iter_cols()` method will return columns:

```
>>> for col in ws.iter_cols(min_row=1, max_col=3, max_row=2):
...     for cell in col:
...         print(cell)
<Cell Sheet1.A1>
<Cell Sheet1.A2>
<Cell Sheet1.B1>
<Cell Sheet1.B2>
<Cell Sheet1.C1>
<Cell Sheet1.C2>
```

tuple（）方法：
将列表转换为元祖

If you need to iterate through all the rows or columns of a file, you can instead use the `openpyxl.worksheet.Worksheet.rows()` property:

```
>>> ws = wb.active
>>> ws['C9'] = 'hello world'
>>> tuple(ws.rows)
((<Cell Sheet.A1>, <Cell Sheet.B1>, <Cell Sheet.C1>),
(<Cell Sheet.A2>, <Cell Sheet.B2>, <Cell Sheet.C2>),
(<Cell Sheet.A3>, <Cell Sheet.B3>, <Cell Sheet.C3>),
(<Cell Sheet.A4>, <Cell Sheet.B4>, <Cell Sheet.C4>),
(<Cell Sheet.A5>, <Cell Sheet.B5>, <Cell Sheet.C5>),
(<Cell Sheet.A6>, <Cell Sheet.B6>, <Cell Sheet.C6>),
(<Cell Sheet.A7>, <Cell Sheet.B7>, <Cell Sheet.C7>),
(<Cell Sheet.A8>, <Cell Sheet.B8>, <Cell Sheet.C8>),
(<Cell Sheet.A9>, <Cell Sheet.B9>, <Cell Sheet.C9>))
```

or the `openpyxl.worksheet.Worksheet.columns()` property:

```
>>> tuple(ws.columns)
((<Cell Sheet.A1>,
<Cell Sheet.A2>,
<Cell Sheet.A3>,
<Cell Sheet.A4>,
<Cell Sheet.A5>,
<Cell Sheet.A6>,
...
<Cell Sheet.B7>,
<Cell Sheet.B8>,
<Cell Sheet.B9>),
(<Cell Sheet.C1>,
<Cell Sheet.C2>,
<Cell Sheet.C3>,
<Cell Sheet.C4>,
<Cell Sheet.C5>,
<Cell Sheet.C6>,
<Cell Sheet.C7>,
<Cell Sheet.C8>,
<Cell Sheet.C9>))
```

Data storage

Once we have a `openpyxl.cell.Cell`, we can assign it a value:

```
>>> c.value = 'hello, world'
>>> print(c.value)
'hello, world'

>>> d.value = 3.14
>>> print(d.value)
3.14
```

You can also enable type and format inference:

```
>>> wb = Workbook(guess_types=True)
>>> c.value = '12%'
>>> print(c.value)
0.12

>>> import datetime
>>> d.value = datetime.datetime.now()
>>> print d.value
datetime.datetime(2010, 9, 10, 22, 25, 18)

>>> c.value = '31.50'
>>> print(c.value)
31.5
```

Saving to a file

The simplest and safest way to save a workbook is by using the `openpyxl.workbook.Workbook.save()` method of the `openpyxl.workbook.Workbook` object:

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

Warning

This operation will overwrite existing files without warning.

Note

Extension is not forced to be xlsx or xlsm, although you might have some trouble opening it directly with another application if you don’t use an official extension.

As OOXML files are basically ZIP files, you can also end the filename with .zip and open it with your favourite ZIP archive manager.

You can specify the attribute *template=True*, to save a workbook as a template:

？模板如何使用？

```
>>> wb = load_workbook('document.xlsx')
>>> wb.template = True
>>> wb.save('document_template.xltx')
```

or set this attribute to *False* (default), to save as a document:

```
>>> wb = load_workbook('document_template.xltx')
>>> wb.template = False
>>> wb.save('document.xlsx', as_template=False)
```

Warning

You should monitor the data attributes and document extensions for saving documents in the document templates and vice versa, otherwise the result table engine can not open the document.

Note

The following will fail:

```
>>> wb = load_workbook('document.xlsx')
>>> # Need to save with the extension *.xlsx
>>> wb.save('new_document.xlsm')
>>> # MS Excel can't open the document
>>>
>>> # or
>>>
>>> # Need specify attribute keep_vba=True
>>> wb = load_workbook('document.xlsm')
>>> wb.save('new_document.xlsm')
>>> # MS Excel will not open the document
>>>
>>> # or
>>>
>>> wb = load_workbook('document.xltm', keep_vba=True)
>>> # If we need a template document, then we must specify extension as *.xltm.
>>> wb.save('new_document.xlsm')
>>> # MS Excel will not open the document
```

Loading from a file

？会针对打开吗？

The same way as writing, you can import `openpyxl.load_workbook()` to open an existing workbook:

```
>>> from openpyxl import load_workbook
>>> wb2 = load_workbook('test.xlsx')
>>> print wb2.get_sheet_names()
['Sheet2', 'New Title', 'Sheet1']
```

This ends the tutorial for now, you can proceed to the Simple usage section