**Python Openpyxl Introduction**

Python provides the Openpyxl module, which is used to deal with Excel files without involving third-party Microsoft application software. By using this module, we can have control over excel without open the application. It is used to perform excel tasks such as read data from excel file, or write data to the excel file, draw some charts, accessing excel sheet, renaming sheet, modification (adding and deleting) in excel sheet, formatting, styling in the sheet, and any other task. Openpyxl is very efficient to perform these tasks for you.

Data scientists often use the Openpyxl to perform different operations such as data copying to data mining as well as data analysis.

**Openpyxl Working Process**

The Openpyxl library is used to write or read the data in the excel file and many other tasks. An excel file that we use for operation is called Workbook that contains a minimum of one Sheet and a maximum of tens of sheets.

Sheets consist of Rows (horizontal series) starting from 1 and Columns (vertical series) starting from A.

Row and column together make a grid and form a cell that may store some data. Data can be of any type, such as numeric, string.

Openpyxl provides flexibility to read data from the individual cell or write data to it.

**Installation of Openpyxl**

In the above section, we have discussed the openpyxl briefly and its working process. In order to use Openpyxl, one should have Python 3.7 and openpyxl 2.6.2 installed in the system. Let's start working with openpyxl by installing openpyxl using the following command:

pip install openpyxl

The xlsx is the extension of the XML spreadsheet file. The xlsx file supports macros. Let's understand the basic operation related to the excel file. Consider the following code:

Example:

import openpyxl

path = "clients.xlsx"

wb = openpyxl.load\_workbook(path)

sh = wb.active

print(sh)

Example:

from openpyxl import Workbook

import time

wb = Workbook()

sheet = wb.active

sheet['A1'] = 87

sheet['A2'] = "Devansh"

sheet['A3'] = 41.80

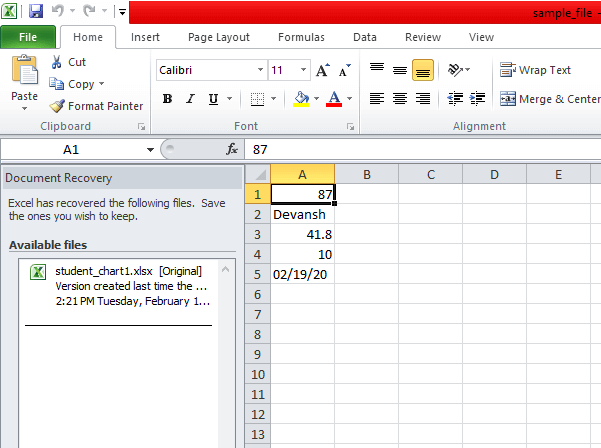
sheet['A4'] = 10

now = time.strftime("%x")

sheet['A5'] = now

wb.save("sample\_file.xlsx")

Output:



In the above code, we have written data into the five cells A1, A2, A3, A4, and A5. These cells consist of different types of values. We have imported Workbook class from the openpyxl module. A workbook class is a container that contains all parts of the document.

Here we have defined a new workbook. At least one sheet is always made with a workbook.

wb = Workbook()

We get the location of the active sheet.

sheet['A1'] = 87

sheet['A2'] = 'Devansh'

We have saved all data to the sample\_file.xlsx file using the save() method.

**Openpyxl Write Data to Cell**

We can add data to the excel file using the following Python code. First, we will import the load\_workbook function from the openpyxl module, then create the object of the file and pass filepath as an argument. Consider the following code:

from openpyxl import load\_workbook

wb = load\_workbook(r'C:\Users\DEVANSH SHARMA\Desktop\demo.xlsx')

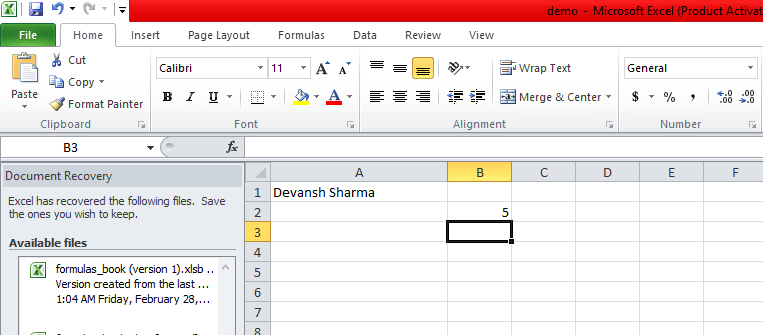
sheet = wb.active

sheet['A1'] = 'Devansh Sharma'

sheet.cell(row=2, column=2).value = 5

wb.save(r'C:\Users\DEVANSH SHARMA\Desktop\demo.xlsx')

Output:



**Openpyxl Append values**

Openpyxl provides an append() method, which is used to append the group of values. We can append any type of value. These values are appended at the bottom of the current working sheet. Consider the following code:

from openpyxl import Workbook

wb = Workbook()

sheet = wb.active

data = (

(11, 48, 50),

(81, 30, 82),

(20, 51, 72),

(21, 14, 60),

(28, 41, 49),

(74, 65, 53),

("Peter", 'Andrew',45.63)

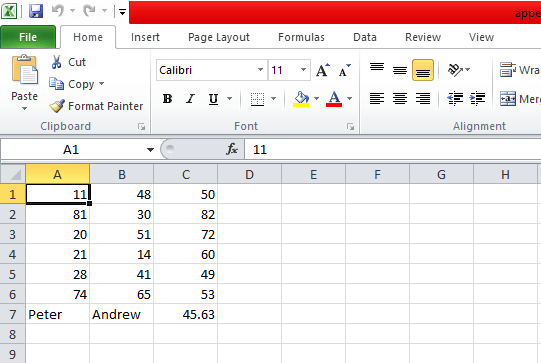
)

for i in data:

sheet.append(i)

wb.save('appending\_values.xlsx')

Output:



**Openpyxl Read Data from cell**

We can read the data that we have previously written in the cell. There are two methods to read a cell, firstly we can access it by cell name, and secondly, we can access it by the cell() function. For example, we are reading the data from the sample\_file.xlrs file.

import openpyxl

wb = openpyxl.load\_workbook('sample\_file.xlsx')

sheet = wb.active

x1 = sheet['A1']

x2 = sheet['A2']

#using cell() function

x3 = sheet.cell(row=3, column=1)

print("The first cell value:",x1.value)

print("The second cell value:",x2.value)

print("The third cell value:",x3.value)

Output:

The first cell value: 87

The second cell value: Devansh

The third cell value: 41.8

**Openpyxl Read multiple cells**

We can read the values from the multiple cells. In the following example, we have marks.xlsx named excel file and we will read each cell of file using the range operator. Let's have a look at the following program:

import openpyxl

wb = openpyxl.load\_workbook('marks.xlsx')

sheet = wb.active

#

cells = sheet['A1','B7']

# cells behave like range operator

for i1,i2 in cells:

print("{0:8} {1:8}".format(i1.value,i2.value))

Output:

Student\_name Marks

Tony Stark 47

Loki 59

Oddin 73

Nick Fury 62

Samaul 75

Peter Parkar 80

**Openpyxl Iterate by rows**

The openpyxl provides the iter\_row() function, which is used to read data corresponding to rows. Consider the following example:

from openpyxl import Workbook

wb = Workbook()

sheet = wb.active

rows = (

(90, 46, 48, 44),

(81, 30, 32, 16),

(23, 95, 87,27),

(65, 12, 89, 53),

(42, 81, 40, 44),

(34, 51, 76, 42)

)

for row in rows:

sheet.append(row)

for row in sheet.iter\_rows(min\_row=1, min\_col=1, max\_row=6, max\_col=4):

for cell in row:

print(cell.value, end=" ")

print()

book.save('iter\_rows.xlsx')

Output:

90 46 48 44

81 30 32 16

23 95 87 27

65 12 89 53

42 81 40 44

34 51 76 42

**Openpyxl Iterate by Column**

The openpyxl provides iter\_col() method which return cells from the worksheet as columns. Consider the following example:

from openpyxl import Workbook

book = Workbook()

sheet = book.active

rows = (

(90, 46, 48, 44),

(81, 30, 32, 16),

(23, 95, 87, 27),

(65, 12, 89, 53),

(42, 81, 40, 44),

(34, 51, 76, 42)

)

for row in rows:

sheet.append(row)

for row in sheet.iter\_cols(min\_row=1, min\_col=1, max\_row=6, max\_col=3):

for cell in row:

print(cell.value, end=" ")

print()

book.save('iterbycols.xlsx')

Output:

90 81 23 65 42 34

46 30 95 12 81 51

48 32 87 89 40 76

**Openpyxl Sheets**

As we know that each workbook can have multiple sheets. First, we need to create more than one sheet in a single workbook then we can access those excel sheets using Python. In the following example, we have created a workbook with three sheets:

import openpyxl

wb = openpyxl.load\_workbook('dimension\_1.xlsx')

#Getting list of all sheet available in workbook

print(wb.get\_sheet\_names())

# Returning object

active\_sheet = wb.active

print(type(active\_sheet))

# Title of sheet

sheet = wb.get\_sheet\_by\_name("Monday")

print(sheet.title)

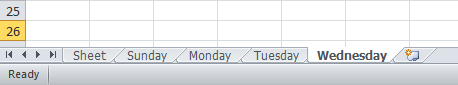
Output:

['Sheet', 'Sunday', 'Monday', 'Tuesday', 'Wednesday']

<class 'openpyxl.worksheet.worksheet.Worksheet'>

Monday

It will look like the following image.



**Openpyxl filter and sort data**

The auto\_filter attribute is used to set filtering and sorting conditions. Consider the following code:

from openpyxl import Workbook

wb = Workbook()

sheet = wb.active

sheet['A3'] = 40

sheet['B3'] = 26

row\_count = [

(93,45),

(23,54),

(80,43),

(21,12),

(63,29),

(34,15),

(80,68),

(20,41)

]

for row in row\_count:

sheet.append(row)

print(sheet.dimensions)

for a1,a2 in sheet[sheet.dimensions]:

print(a1.value, a2.value)

sheet.auto\_filter.add\_sort\_condition('B2:B8')

sheet.auto\_filter.add\_filter\_column(1, ['40', '26'])

wb.save('dimension\_1.xlsx')

Output:

A3:B11

40 26

93 45

23 54

80 43

21 12

63 29

34 15

80 68

20 41

Openpyxl Merging cell

We can merge the cell using the merge\_cells() method. When we merge the cells, the top-left one is removed from the worksheet. The openpyxl also provides the unmerged\_cells() method to unmerge the cell. Consider the following code:

from openpyxl.styles import Alignment

wb = Workbook()

sheet = wb.active

sheet.merge\_cells('A1:B2')

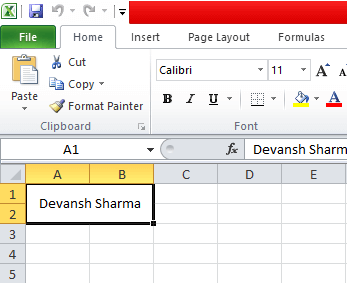
cell = sheet.cell(row=1, column=1)

cell.value = 'Devansh Sharma'

cell.alignment = Alignment(horizontal='center', vertical='center')

wb.save('merging.xlsx')

Output:



The freezing panes are simply mean to freeze an area of worksheet as visible while scrolling to other parts of the worksheet. It is useful feature where we can display the top row or leftmost column on the screen. We can do this by passing the cell name to the freeze\_panes variable. To unfreeze all panes, set freeze\_panes to 'None'. Consider the following code:

from openpyxl import Workbook

from openpyxl.styles import Alignment

wb = Workbook()

sheet = wb.active

sheet.freeze\_panes = 'A1'

wb.save('freez\_cells.xlsx')

Output:

Run the above code and scroll the worksheet.

Openpyxl formulas

We can write formula into the cell. These formulas are used to perform operations in excel file. After writing in the cell execute it from the workbook. Consider the following example:

from openpyxl import Workbook

wb = Workbook()

sheet = wb.active

rows\_count = (

(14, 27),

(22, 30),

(42, 92),

(51, 32),

(16, 60),

(63, 13)

)

for i in rows\_count:

sheet.append(i)

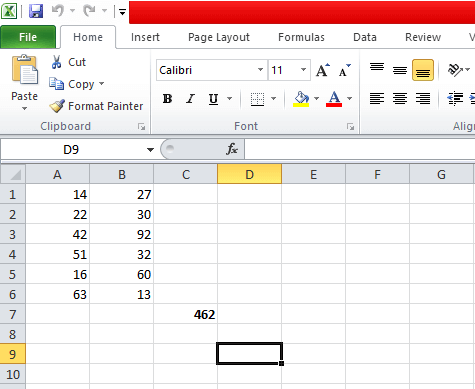
cell = sheet.cell(row=7, column=3)

cell.value = "=SUM(A1:B6)"

cell.font = cell.font.copy(bold=True)

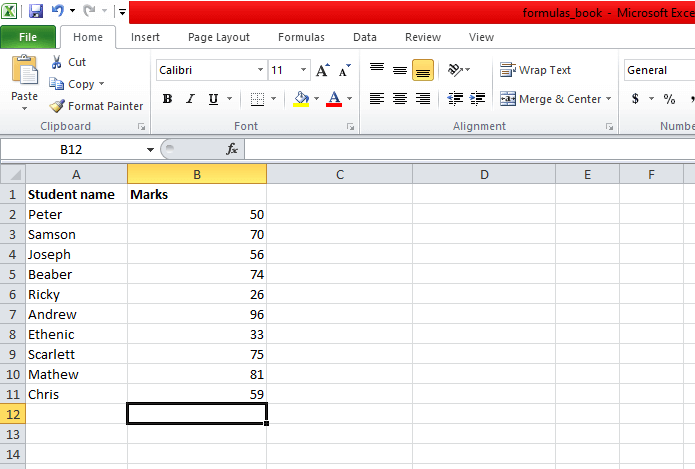
wb.save('formulas\_book.xlsx')

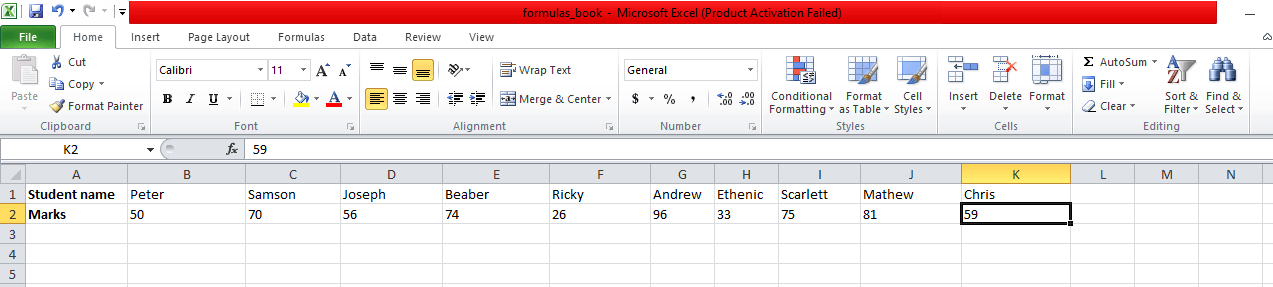
Output:



Openpyxl Cell Inverter

The **openpyxl** cell inverter is used to invert the row and column of the cell in the spreadsheet. For example, the value at 3 rows will be inverted into 5 columns and row at row 5 will invert to column 3 (vice versa). You can see in the following images





This program is written with the help of nested for loop. First the data structure writes in the sheetData[x][y] for the cell at column x and row y then the newly created spreadsheet in the spreadData[y][x] for the cell at column y and row x.

Adding Chart to Excel File

Charts are effective way to represent the data. Using the charts, it visualizes the data which can be easily untestable. There are various types of chart: pie chart, line chart, bar chart, and so on. We can draw a chart on a spreadsheet using an openpyxl module.

For building any chart on the spreadsheet, we need to define the chart type like BarChart, LineChart, and so on. We also import reference which represents data that is used for the chart. It is important to define what data we want to represent on the chart. Let's understand by the following example:

from openpyxl import Workbook

from openpyxl.chart import BarChart, Reference

wb = Workbook()

sheet = wb.active

# Let's create some sample student data

rows = [

["Serial\_no", "Roll no", "Marks"],

[1, "0090011", 75],

[2, "0090012", 60],

[3, "0090013", 43],

[4, "0090014", 97],

[5, "0090015", 63],

[6, "0090016", 54],

[7, "0090017", 86],

]

for i in rows:

sheet.append(i)

chart = BarChart()

values = Reference(worksheet=sheet,

min\_row=1,

max\_row=8,

min\_col=2,

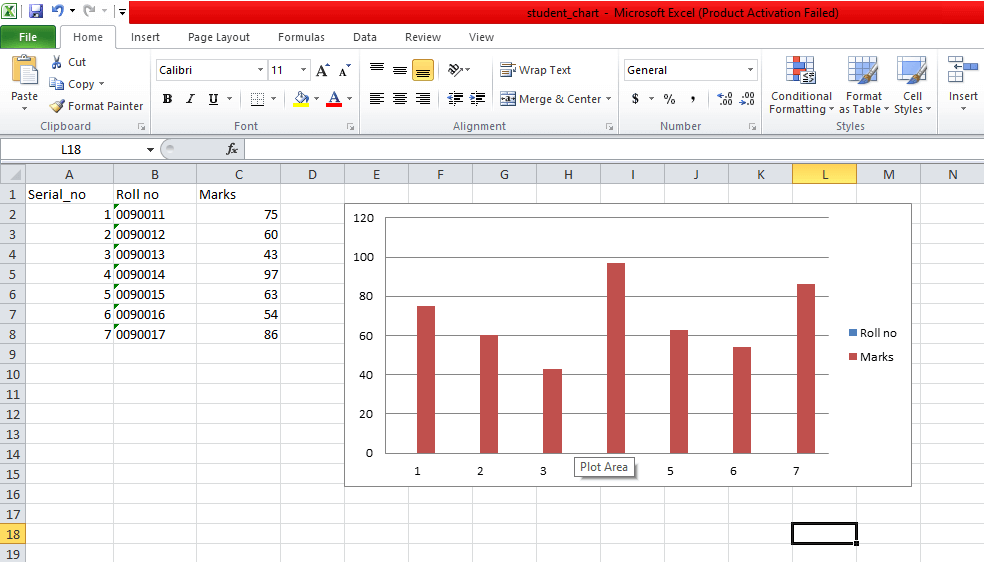
max\_col=3)

chart.add\_data(values, titles\_from\_data=True)

sheet.add\_chart(chart, "E2")

wb.save("student\_chart.xlsx")

Output:



In the above code, we have created the sample data and drawn the bar chart corresponding to sample data.

Now we will create the line chart. Consider the following code:

import random

from openpyxl import Workbook

from openpyxl.chart import LineChart, Reference

wb = Workbook()

spreadsheet = wb.active

# Let's create some sample data

rows = [

["", "January", "February", "March", "April",

"May", "June", "July", "August", "September",

"October", "November", "December"],

[1, ],

[2, ],

[3, ],

]

for row in rows:

spreadsheet.append(row)

for row in spreadsheet.iter\_rows(min\_row=2,

max\_row=4,

min\_col=2,

max\_col=13):

for cell in row:

cell.value = random.randrange(5, 100)

chart = LineChart()

data = Reference(worksheet=spreadsheet,

min\_row=2,

max\_row=4,

min\_col=1,

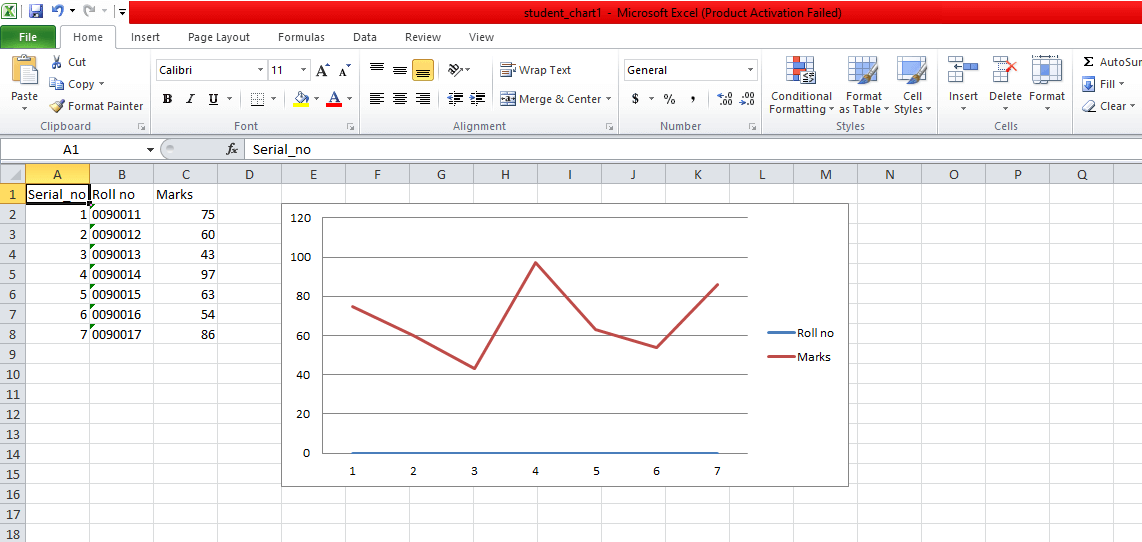
max\_col=13)

chart.add\_data(data, from\_rows=True, titles\_from\_data=True)

spreadsheet.add\_chart(chart, "C6")

wb.save("line\_chart1.xlsx")

Output:



In the above code, we are used from\_rows = True as a parameter, it denotes chart plot row by row instead of the column by column.

Adding Image

Images are not generally used in a spreadsheet but sometimes we can use as per our requirement. We can use an image for the branding purposes or to make the spreadsheet more personal and attractive. For loading an image to spreadsheet, we need to install an additional module called pillow by the following command.

pip install pillow

In the following program, we are importing the image into the excel file.

from openpyxl import load\_workbook

from openpyxl.drawing.image import Image

# Let's use the hello\_world spreadsheet since it has less data

workbook = load\_workbook(filename="student\_chart1.xlsx")

sheet = workbook.active

logo = Image(r"C:\Users\DEVANSH SHARMA\Pictures\Screenshots\image.png")

# A bit of resizing to not fill the whole spreadsheet with the logo

logo.height = 150

logo.width = 150

sheet.add\_image(logo, "E2")

workbook.save(filename="hello\_world\_logo1.xlsx")

-------------------------------------------------------------------------------------------------------------------------------------------------------

**Tutorial**

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 work:

>>> from openpyxl import Workbook

>>> wb = Workbook()

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

>>> ws = wb.active

Note

This is set to 0 by default. Unless you modify its value, you will always get the first worksheet by using this method.

You can create new worksheets using the 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

# or

>>> ws3 = wb.create\_sheet("Mysheet", -1) # insert at the penultimate 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 Worksheet.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 Worksheet.sheet\_properties.tabColor attribute:

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 Workbook.sheetname attribute

>>> print(wb.sheetnames)

['Sheet2', 'New Title', 'Sheet1']

You can loop through worksheets

>>> for sheet in wb:

... print(sheet.title)

You can create copies of worksheets within a single workbook:

Workbook.copy\_worksheet() method:

>>> source = wb.active

>>> target = wb.copy\_worksheet(source)

Note

Only cells (including values, styles, hyperlinks and comments) and certain worksheet attribues (including dimensions, format and properties) are copied. All other workbook / worksheet attributes are not copied - e.g. Images, Charts.

You also cannot copy worksheets between workbooks. You cannot copy a worksheet if the workbook is open in read-only or write-only mode.

Playing with data

Accessing one cell

Now we know how to get 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 Worksheet.cell() method.

This provides access to cells using row and column notation:

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

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 x in range(1,101):

... for y in range(1,101):

... ws.cell(row=x, column=y)

will create 100x100 cells in memory, for nothing.

Accessing many cells

Ranges of cells can be accessed using slicing:

>>> cell\_range = ws['A1':'C2']

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 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 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>

Note

For performance reasons the Worksheet.iter\_cols() method is not available in read-only mode.

If you need to iterate through all the rows or columns of a file, you can instead use the 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 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>))

Note

For performance reasons the Worksheet.columns property is not available in read-only mode.

Values only

If you just want the values from a worksheet you can use the Worksheet.values property. This iterates over all the rows in a worksheet but returns just the cell values:

for row in ws.values:

for value in row:

print(value)

Both Worksheet.iter\_rows() and Worksheet.iter\_cols() can take the values\_only parameter to return just the cell’s value:

>>> for row in ws.iter\_rows(min\_row=1, max\_col=3, max\_row=2, values\_only=True):

... print(row)

(None, None, None)

(None, None, None)

Data storage

Once we have a 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

Saving to a file

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

>>> wb = Workbook()

>>> wb.save('balances.xlsx')

Warning

This operation will overwrite existing files without warning.

Note

The filename 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 open it with your favourite ZIP archive manager.

Saving as a stream

If you want to save the file to a stream, e.g. when using a web application such as Pyramid, Flask or Django then you can simply provide a NamedTemporaryFile():

>>> from tempfile import NamedTemporaryFile

>>> from openpyxl import Workbook

>>> wb = Workbook()

>>> with NamedTemporaryFile() as tmp:

wb.save(tmp.name)

tmp.seek(0)

stream = tmp.read()

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

>>> 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

The same way as writing, you can use the openpyxl.load\_workbook() to open an existing workbook:

>>> from openpyxl import load\_workbook

>>> wb2 = load\_workbook('test.xlsx')

>>> print(wb2.sheetnames)

['Sheet2', 'New Title', 'Sheet1']

Simple usage

Write a workbook

>>> from openpyxl import Workbook

>>> from openpyxl.utils import get\_column\_letter

>>>

>>> wb = Workbook()

>>>

>>> dest\_filename = 'empty\_book.xlsx'

>>>

>>> ws1 = wb.active

>>> ws1.title = "range names"

>>>

>>> for row in range(1, 40):

... ws1.append(range(600))

>>>

>>> ws2 = wb.create\_sheet(title="Pi")

>>>

>>> ws2['F5'] = 3.14

>>>

>>> ws3 = wb.create\_sheet(title="Data")

>>> for row in range(10, 20):

... for col in range(27, 54):

... \_ = ws3.cell(column=col, row=row, value="{0}".format(get\_column\_letter(col)))

>>> print(ws3['AA10'].value)

AA

>>> wb.save(filename = dest\_filename)

Read an existing workbook

>>> from openpyxl import load\_workbook

>>> wb = load\_workbook(filename = 'empty\_book.xlsx')

>>> sheet\_ranges = wb['range names']

>>> print(sheet\_ranges['D18'].value)

3

Note

There are several flags that can be used in load\_workbook.

data\_only controls whether cells with formulae have either the formula (default) or the value stored the last time Excel read the sheet.

keep\_vba controls whether any Visual Basic elements are preserved or not (default). If they are preserved they are still not editable.

Warning

openpyxl does currently not read all possible items in an Excel file so images and charts will be lost from existing files if they are opened and saved with the same name.

Using number formats

>>> import datetime

>>> from openpyxl import Workbook

>>> wb = Workbook()

>>> ws = wb.active

>>> # set date using a Python datetime

>>> ws['A1'] = datetime.datetime(2010, 7, 21)

>>>

>>> ws['A1'].number\_format

'yyyy-mm-dd h:mm:ss'

Using formulae

>>> from openpyxl import Workbook

>>> wb = Workbook()

>>> ws = wb.active

>>> # add a simple formula

>>> ws["A1"] = "=SUM(1, 1)"

>>> wb.save("formula.xlsx")

Warning

NB you must use the English name for a function and function arguments must be separated by commas and not other punctuation such as semi-colons.

openpyxl never evaluates formula but it is possible to check the name of a formula:

>>> from openpyxl.utils import FORMULAE

>>> "HEX2DEC" in FORMULAE

True

If you’re trying to use a formula that isn’t known this could be because you’re using a formula that was not included in the initial specification. Such formulae must be prefixed with \_xlfn. to work.

Merge / Unmerge cells

When you merge cells all cells but the top-left one are removed from the worksheet. To carry the border-information of the merged cell, the boundary cells of the merged cell are created as MergeCells which always have the value None. See Styling Merged Cells for information on formatting merged cells.

>>> from openpyxl.workbook import Workbook

>>>

>>> wb = Workbook()

>>> ws = wb.active

>>>

>>> ws.merge\_cells('A2:D2')

>>> ws.unmerge\_cells('A2:D2')

>>>

>>> # or equivalently

>>> ws.merge\_cells(start\_row=2, start\_column=1, end\_row=4, end\_column=4)

>>> ws.unmerge\_cells(start\_row=2, start\_column=1, end\_row=4, end\_column=4)

Inserting an image

>>> from openpyxl import Workbook

>>> from openpyxl.drawing.image import Image

>>>

>>> wb = Workbook()

>>> ws = wb.active

>>> ws['A1'] = 'You should see three logos below'

>>> # create an image

>>> img = Image('logo.png')

>>> # add to worksheet and anchor next to cells

>>> ws.add\_image(img, 'A1')

>>> wb.save('logo.xlsx')

Fold (outline)

>>> import openpyxl

>>> wb = openpyxl.Workbook()

>>> ws = wb.create\_sheet()

>>> ws.column\_dimensions.group('A','D', hidden=True)

>>> ws.row\_dimensions.group(1,10, hidden=True)

>>> wb.save('group.xlsx')