Chapter 21
Working with Excel Files

21.1 Introduction

This chapter introduces the `openpyxl` module that can be used when working with Excel files. Excel is a software application developed by Microsoft that allows users to work with spreadsheets. It is a very widely used tool and files using the Excel file format are commonly encountered within many organisations. It is in effect the industry standard for spreadsheets and as such is a very useful tool to have in the developers toolbox.

21.2 Excel Files

Although CSV files are a convenient and simple way to handle data; it is very common to need to be able to read or write Excel files directly. To this end there are several libraries available in Python for this purpose. One widely used library is the *OpenPyXL* library. This library was originally written to support access to Excel 2010 files. It is an open source project and is well documented.

The OpenPyXL library provides facilities for

- reading and writing Excel workbooks,
- creating/accessing Excel worksheets,
- creating Excel formulas,
- creating graphs (with support from additional modules).

As *OpenPyXL* is not part of the standard Python distribution you will need to install the library yourself using a tool such as Anaconda or pip (e.g. `pip install openpyxl`). Alternatively, if you are using PyCharm you will be able to add the OpenPyXL library to your project.

J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_21

21.3 The Openpyxl. Workbook Class

The key element in the OpenPyXL library is the Workbook class. This can be imported from the module:

```
from openpyxl import Workbook
```

A new instance of the (in memory) Workbook can be created using the Workbook class (note at this point it is purely a structure within the Python program and must be saved before an actual Excel file is created).

```
wb = Workbook()
```

21.4 The Openpyxl. WorkSheet Objects

A workbook is always created with at least one worksheet. You can get hold of the currently active worksheet using the `Workbook.active` property:

```
ws = wb.active
```

You can create additional worksheets using the workbooks' `create_sheet()` method:

```
ws = wb.create_sheet('Mysheet')
```

You can access or update the title of the worksheet using the `title` property:

```
ws.title = 'New Title'
```

The background colour of the tab holding this title is white by default. You can change this providing an RRGGBB colour code to the `worksheet.sheet_properties.tabColor` attribute, for example:

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

21.5 Working with Cells

It is possible to access the cells within a worksheet. A cell can be accessed directly as keys on the worksheet, for example:

```
ws['A1'] = 42
```

or

```
cell = ws['A1']
```

This returns a cell object; you can obtain the value of the cell using the value property, for example

```
print(cell.value)
```

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)
```

A row of values can also be added at the current position within the Excel file using append:

```
ws.append([1, 2, 3])
```

This will add a row to the Excel file containing 1, 2, and 3.
Ranges of cells can be accessed using slicing:

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

Ranges of rows or columns can also be obtained:

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

The value of a cell can also be an Excel formula such as

```
ws['A3'] = '=SUM(A1, A2)'
```

A workbook is actually only a structure in memory; it must be saved to a file for permanent storage. These workbooks can be saved using the `save()` method. This method takes a filename and writes the `Workbook` out in Excel format.

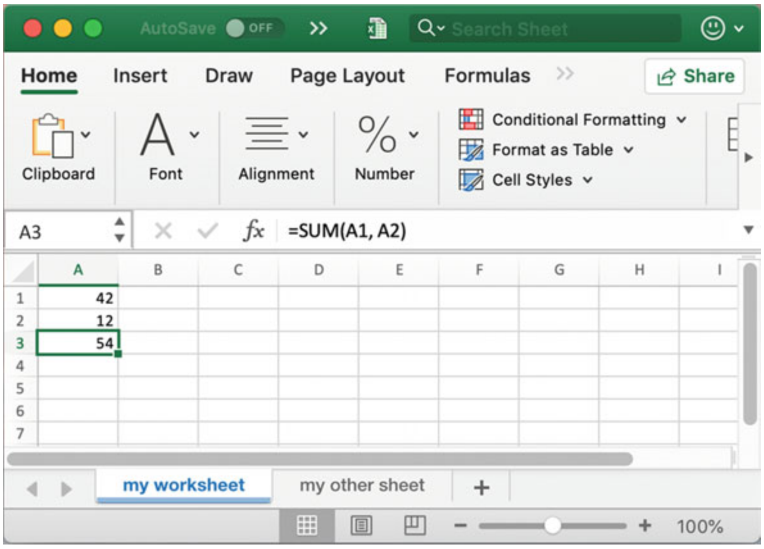
```
workbook = Workbook()
...
workbook.save('balances.xlsx')
```

21.6 Sample Excel File Creation Application

The following simple application creates a `Workbook` with two worksheets. It also contains a simple Excel formula that sums the values held in to other cells:

```
from openpyxl import Workbook

def main():
    print('Starting Write Excel Example with openPyXL')

    workbook = Workbook()
    # Get the current active worksheet
    ws = workbook.active
    ws.title = 'my worksheet'
    ws.sheet_properties.tabColor = '1072BA'

    ws['A1'] = 42
    ws['A2'] = 12
    ws['A3'] = '=SUM(A1, A2)'

    ws2 = workbook.create_sheet(title='my other sheet')
    ws2['A1'] = 3.42
    ws2.append([1, 2, 3])
    ws2.cell(column=2, row=1, value=15)

    workbook.save('sample.xlsx')

    print('Done Write Excel Example')

if __name__ == '__main__':
    main()
```

The Excel file generated from this can be viewed in Excel as shown below:



A3 =SUM(A1, A2)

	A	B	C	D	E	F	G	H	I
1	42								
2	12								
3	54								
4									
5									
6									
7									

21.7 Loading a Workbook from an Excel File

Of course, in many cases it is necessary not just to create Excel files for data export but also to import data from an existing Excel file. This can be done using the OpenPyXL `load_workbook()` function. This function opens the specified Excel file (in read only mode by default) and returns a Workbook object.

```
from openpyxl import load_workbook

workbook = load_workbook(filename='sample.xlsx')
```

You can now access a list of sheets, their names, obtain the currently active sheet etc. using properties provided by the `workbook` object:

- `workbook.active` returns the active worksheet object.
- `workbook.sheetnames` returns the names (strings) of the worksheets in this workbook.
- `workbook.worksheets` returns a list of worksheet objects.

The following sample application reads the Excel file created earlier in this chapter:

```
from openpyxl import load_workbook

def main():
    print('Starting reading Excel file using openPyXL')

    workbook = load_workbook(filename='sample.xlsx')
    print(workbook.active)
    print(workbook.sheetnames)
    print(workbook.worksheets)

    print('-' * 10)
    ws = workbook['my worksheet']
    print(ws['A1'])
    print(ws['A1'].value)
    print(ws['A2'].value)
    print(ws['A3'].value)

    print('-' * 10)
    for sheet in workbook:
        print(sheet.title)

    print('-' * 10)
    cell_range = ws['A1':'A3']
    for cell in cell_range:
        print(cell[0].value)
    print('-' * 10)
    print('Finished reading Excel file using openPyXL')

if __name__ == '__main__':
    main()
```

The output from this application is illustrated below:

```
Starting reading Excel file using openPyXL
<Worksheet "my worksheet">
['my worksheet', 'my other sheet']
[<Worksheet "my worksheet">, <Worksheet "my other sheet">]
----------
<Cell 'my worksheet'.A1>
42
12
=SUM(A1, A2)
----------
my worksheet
my other sheet
----------
42
12
=SUM(A1, A2)
----------
Finished reading Excel file using openPyXL
```

21.8 Online Resources

See the following online resources for information on the topics in this chapter:

- https://openpyxl.readthedocs.io/en/stable for documentation on the OpenPyXL Python to Excel library.

21.9 Exercises

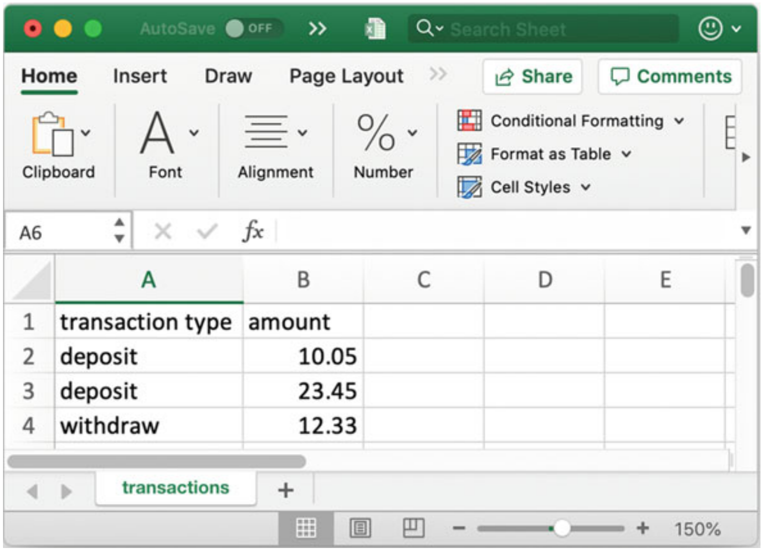
Using the Account class that you created in the last chapter; write the account transaction information to an Excel file instead of a CSV file.

To do this create a function called `write_account_transaction_to_excel()` that takes the name of the Excel file and the account to store. The function should then write the data to the file using the excel format.

The following sample application illustrates how this function might be used:

```
print('Starting')
acc = accounts.CurrentAccount('123', 'John', 10.05, 100.0)
acc.deposit(23.45)
acc.withdraw(12.33)

print('Writing Account Transactions')
write_account_transaction_to_excel('accounts.xlsx', acc)

print('Done')
```

The contents of the Excel file would then be:



	A	B	C	D	E
1	transaction type	amount			
2	deposit	10.05			
3	deposit	23.45			
4	withdraw	12.33			