Automating Excel Tasks with OpenPyXL

`openpyxl` is a Python library used to work with Excel files (xlsx format). You can use it to automate common Excel tasks, such as reading and writing data, formatting cells, creating charts, and more. Here's a guide to help you get started with automating Excel tasks using `openpyxl`.

---

1. Installing OpenPyXL

You need to install `openpyxl` if you haven't already:

bash
Copy code
```
pip install openpyxl
```

---

2. Working with Excel Files

Loading an Existing Workbook

To load an existing Excel file:

python
Copy code
```
from openpyxl import load_workbook

# Load the workbook
wb = load_workbook('example.xlsx')

# Get a specific sheet
sheet = wb['Sheet1']  # Or wb.active for the active sheet
```

Creating a New Workbook

To create a new workbook:

python
Copy code
```
from openpyxl import Workbook

# Create a new workbook
wb = Workbook()
```

```
# Select the active sheet (by default, the first sheet)
sheet = wb.active
```

---

3. Reading Data from Excel

You can read data from specific cells or ranges of cells in the Excel sheet.

Accessing a Single Cell

python
Copy code

```
# Read the value of a specific cell
value = sheet['A1'].value
print(value)
```

Accessing a Range of Cells

python
Copy code

```
# Read a range of cells (e.g., cells A1 to C3)
for row in sheet['A1:C3']:
    for cell in row:
        print(cell.value)
```

Accessing All Rows and Columns

python
Copy code

```
# Loop through all rows
for row in sheet.iter_rows():
    for cell in row:
        print(cell.value)

# Loop through all columns
for col in sheet.iter_cols():
    for cell in col:
        print(cell.value)
```

---

4. Writing Data to Excel

You can easily write data into specific cells or ranges of cells.

Writing to a Single Cell

python
Copy code

```
# Write data to a specific cell
sheet['A1'] = 'Hello, World!'
sheet['B2'] = 42

# Save the workbook
wb.save('example.xlsx')
```

Writing Data to a Range of Cells

python
Copy code

```
# Write data to a range of cells (e.g., A1 to C3)
data = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for i, row in enumerate(data, start=1):
    for j, value in enumerate(row, start=1):
        sheet.cell(row=i, column=j, value=value)

# Save the workbook
wb.save('example.xlsx')
```

---

5. Formatting Cells

OpenPyXL allows you to format cells, such as changing fonts, adding colors, adjusting borders, and more.

Changing Font Style

python
Copy code

```
from openpyxl.styles import Font

# Change font to bold
sheet['A1'].font = Font(bold=True)

# Change font size and color
```

```
sheet['B2'].font = Font(size=14, color="FF0000")  # Red font color

wb.save('example.xlsx')
```

Changing Cell Color

python
Copy code

```
from openpyxl.styles import PatternFill

# Apply a background color (e.g., light yellow)
sheet['A1'].fill = PatternFill(start_color="FFFF00",
end_color="FFFF00", fill_type="solid")

wb.save('example.xlsx')
```

Adjusting Borders

python
Copy code

```
from openpyxl.styles import Border, Side

# Add borders to a cell
border = Border(
    left=Side(border_style="thin", color="000000"),
    right=Side(border_style="thin", color="000000"),
    top=Side(border_style="thin", color="000000"),
    bottom=Side(border_style="thin", color="000000")
)

sheet['A1'].border = border

wb.save('example.xlsx')
```

---

6. Working with Formulas

OpenPyXL allows you to write formulas into cells, which will be calculated when the file is opened in Excel.

python
Copy code

```
# Write a simple formula (e.g., SUM function)
```

```
sheet['A1'] = 10
sheet['A2'] = 20
sheet['A3'] = "=SUM(A1:A2)"  # Formula to sum A1 and A2

wb.save('example.xlsx')
```

---

7. Creating and Modifying Sheets

You can create, rename, and remove sheets in your workbook.

Adding a New Sheet
python
Copy code
```
# Create a new sheet
wb.create_sheet('NewSheet')

# Save the workbook
wb.save('example.xlsx')
```

Renaming a Sheet
python
Copy code
```
# Rename the sheet
sheet.title = 'RenamedSheet'

# Save the workbook
wb.save('example.xlsx')
```

Deleting a Sheet
python
Copy code
```
# Delete a sheet
del wb['RenamedSheet']

# Save the workbook
wb.save('example.xlsx')
```

---

8. Handling Excel Tables and Charts

Creating a Table

python
Copy code

```
from openpyxl.worksheet.table import Table, TableStyleInfo

# Define the range of the table
table = Table(displayName="MyTable", ref="A1:C3")

# Add the table to the sheet
sheet.add_table(table)

# Apply some styling to the table
style = TableStyleInfo(
    name="TableStyleMedium9", showFirstColumn=False,
showLastColumn=False,
    showRowStripes=True, showColumnStripes=True)
table.tableStyleInfo = style

wb.save('example.xlsx')
```

Creating a Chart

python
Copy code

```
from openpyxl.chart import BarChart, Reference

# Create a Bar chart
chart = BarChart()
data = Reference(sheet, min_col=2, min_row=1, max_col=3, max_row=3)
chart.add_data(data, titles_from_data=True)

# Add the chart to the sheet
sheet.add_chart(chart, "E5")

wb.save('example.xlsx')
```

---

9. Automating Excel Tasks in a Loop

You can use loops to automate repetitive tasks, such as populating large datasets or updating cells.

python

Copy code

```
# Automate the population of rows with data
for row in range(1, 101):
    sheet[f'A{row}'] = row  # Writing row number to column A

wb.save('example.xlsx')
```

---

10. Saving and Closing the Workbook

After performing your operations, always save the workbook to persist the changes.

python
Copy code

```
wb.save('example.xlsx')  # Save the workbook

# Optional: Close the workbook (not necessary in most cases)
wb.close()
```

Summary of Key OpenPyXL Features

1. **Reading and Writing**:
    - `load_workbook()` to read an Excel file.
    - `sheet.cell()` to access specific cells.
    - `.value` to read/write cell content.
2. **Formatting**:
    - Apply fonts, colors, and borders to cells.
    - Use `openpyxl.styles` for styling cells.
3. **Formulas**:
    - Write Excel formulas into cells (e.g., SUM, AVERAGE).
4. **Sheets**:
    - Add, rename, and delete sheets in a workbook.
5. **Tables and Charts**:
    - Create Excel tables and add charts (e.g., bar charts, line charts).
6. **Automation**:
    - Automate repetitive tasks using loops and conditional logic.

`openpyxl` is a versatile library for automating Excel tasks. With it, you can handle everything from basic data input/output to advanced formatting, formulas, and charting.