**Working with Excel Spreadsheets in Python**

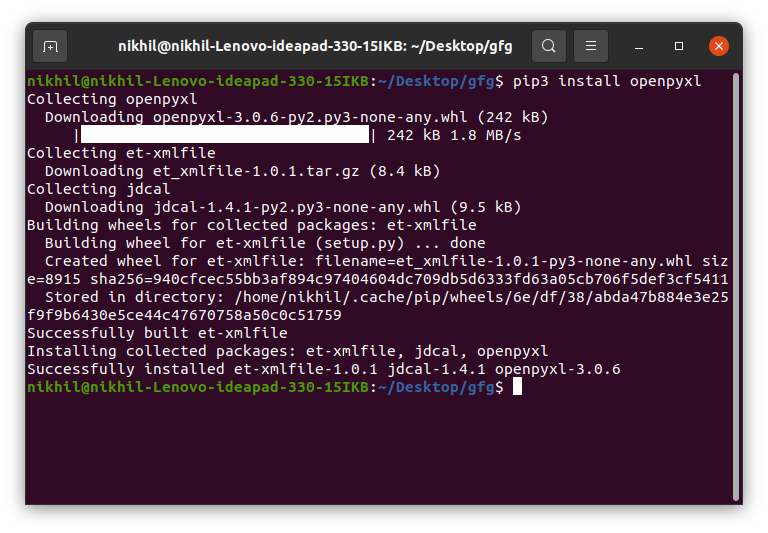
Last Updated : 21 Aug, 2024

You all must have worked with Excel at some time in your life and must have felt the need to automate some repetitive or tedious task. Don’t worry in this tutorial we are going to learn about how to work with Excel using Python, or automating Excel using Python. We will be covering this with the help of the Openpyxl module and will also see how to get [Python](https://www.geeksforgeeks.org/python-programming-language/) in Excel.

**Getting Started Python Openpyxl**

**Openpyxl**is a Python library that provides various methods to interact with Excel Files using Python. It allows operations like reading, writing, arithmetic operations, plotting graphs, etc. This module does not come in-built with Python. To install this type the below command in the terminal.

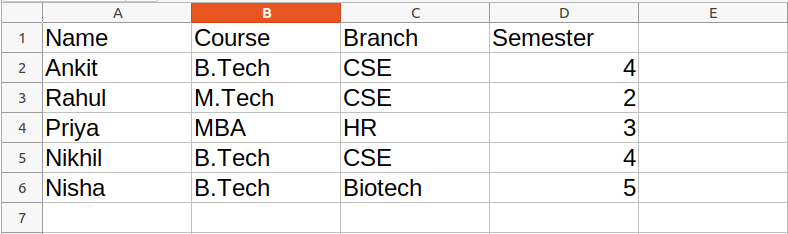
pip install openpyxl



**Read an Excel File in Python**

To read an Excel file you have to open the spreadsheet using the **load\_workbook()** method. After that, you can use the **active**to select the first sheet available and the **cell** attribute to select the cell by passing the row and column parameter. The **value** attribute prints the value of the particular cell. See the below example to get a better understanding.

**Note:**The first row or column integer is 1, not 0.



**Example:**

In this example, a Python program uses the openpyxl module to read an Excel file (“gfg.xlsx”), opens the workbook, and retrieves the value of the cell in the first row and first column, printing it to the console.

*# import openpyxl module*

**import** **openpyxl**

*# Give the location of the file*

path = "gfg.xlsx"

*# To open the workbook*

*# workbook object is created*

wb\_obj = openpyxl.load\_workbook(path)

*# Get workbook active sheet object*

*# from the active attribute*

sheet\_obj = wb\_obj.active

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

print(cell\_obj.value)

**Output:**

Name

**Python Openpyxl Read multiple cells**

There can be two ways of reading from multiple cells:

* Reading through Rows and Columns in Excel with openpyxl
* Read from multiple cells using the cell name

**Reading through Rows and Columns in Excel with openpyxl**

We can get the count of the total rows and columns using the **max\_row** and **max\_column** respectively. We can use these values inside the for loop to get the value of the desired row or column or any cell depending upon the situation. Let’s see how to get the value of the first column and first row.

In this example, a Python program using the openpyxl module reads an Excel file (“gfg.xlsx”). It retrieves and prints the total number of rows and columns in the active sheet, followed by displaying the values of the first column and first row through iterative loops.

**import** **openpyxl**

*# Give the location of the file*

path = "gfg.xlsx"

wb\_obj = openpyxl.load\_workbook(path)

sheet\_obj = wb\_obj.active

row = sheet\_obj.max\_row

column = sheet\_obj.max\_column

print("Total Rows:", row)

print("Total Columns:", column)

print("**\n**Value of first column")

**for** i **in** range(1, row + 1):

cell\_obj = sheet\_obj.cell(row=i, column=1)

print(cell\_obj.value)

print("**\n**Value of first row")

**for** i **in** range(1, column + 1):

cell\_obj = sheet\_obj.cell(row=2, column=i)

print(cell\_obj.value, end=" ")

**Output:**

Total Rows: 6  
Total Columns: 4  
Value of first column  
Name  
Ankit  
Rahul  
Priya  
Nikhil  
Nisha  
Value of first row  
Ankit B.Tech CSE 4

**Read from Multiple Cells Using the Cell Name**

We can also read from multiple cells using the cell name. This can be seen as the list slicing of Python. In this example, a Python program utilizes the openpyxl module to read an Excel file (“gfg.xlsx”). It creates a cell object by specifying a range from ‘A1’ to ‘B6’ in the active sheet and prints the values of each cell pair within that range using a for loop.

**import** **openpyxl**

*# Give the location of the file*

path = "gfg.xlsx"

wb\_obj = openpyxl.load\_workbook(path)

sheet\_obj = wb\_obj.active

cell\_obj = sheet\_obj['A1': 'B6']

**for** cell1, cell2 **in** cell\_obj:

print(cell1.value, cell2.value)

**Output:**

Name Course  
Ankit B.Tech  
Rahul M.Tech  
Priya MBA  
Nikhil B.Tech  
Nisha B.Tech

Refer to the below article to get detailed information about reading excel files using openpyxl.

* [Reading an excel file using Python openpyxl module](https://www.geeksforgeeks.org/python-reading-excel-file-using-openpyxl-module/)

**Python Write Excel File**

First, let’s create a new spreadsheet, and then we will write some data to the newly created file. An empty spreadsheet can be created using the **Workbook()**method. Let’s see the below example.

**Example:**

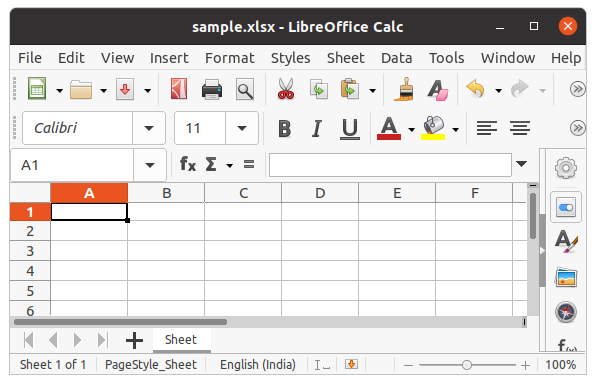
In this example, a new blank Excel workbook is generated using the openpyxl library’s Workbook() function, and it is saved as “sample.xlsx” with the save() method. This code demonstrates the fundamental steps for creating and saving an Excel file in Python.

**from** **openpyxl** **import** Workbook

workbook = Workbook()

workbook.save(filename="sample.xlsx")

**Output:**



After creating an empty file, let’s see how to add some data to it using Python. To add data first we need to select the active sheet and then using the cell() method we can select any particular cell by passing the row and column number as its parameter. We can also write using cell names. See the below example for a better understanding.

**Example:**

In this example, the openpyxl module is used to create a new Excel workbook and populate cells with values such as “Hello,” “World,” “Welcome,” and “Everyone.” The workbook is then saved as “sample.xlsx,” illustrating the process of writing data to specific cells and saving the changes

*# import openpyxl module*

**import** **openpyxl**

wb = openpyxl.Workbook()

sheet = wb.active

c1 = sheet.cell(row=1, column=1)

*# writing values to cells*

c1.value = "Hello"

c2 = sheet.cell(row=1, column=2)

c2.value = "World"

c3 = sheet['A2']

c3.value = "Welcome"

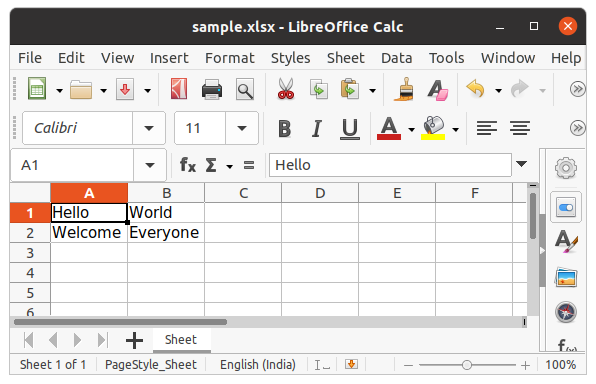
*# B2 means column = 2 & row = 2.*

c4 = sheet['B2']

c4.value = "Everyone"

wb.save("sample.xlsx")

**Output:**



Refer to the below article to get detailed information about writing to excel.

* [Writing to an excel file using openpyxl module](https://www.geeksforgeeks.org/python-writing-excel-file-using-openpyxl-module/)

**Append data in excel using Python**

In the above example, you will see that every time you try to write to a spreadsheet the existing data gets overwritten, and the file is saved as a new file. This happens because the **Workbook()**method always creates a new workbook file object. To write to an existing workbook you must open the file with the **load\_workbook()** method. We will use the above-created workbook.

**Example:**

In this example, the openpyxl module is employed to load an existing Excel workbook (“sample.xlsx”). The program accesses cell ‘A3’ in the active sheet, updates its value to “New Data,” and then saves the modified workbook back to “sample.xlsx.”

*# import openpyxl module*

**import** **openpyxl**

wb = openpyxl.load\_workbook("sample.xlsx")

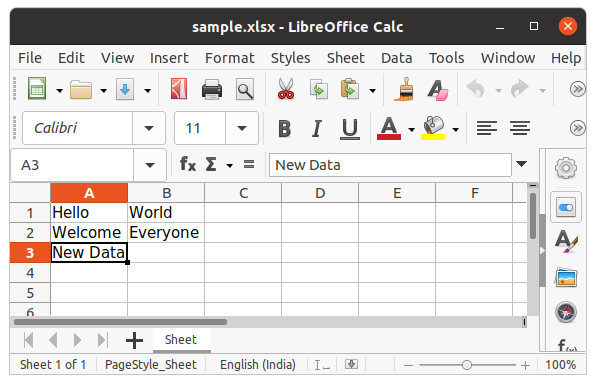
sheet = wb.active

c = sheet['A3']

c.value = "New Data"

wb.save("sample.xlsx")

**Output:**



We can also use the **append()**method to append multiple data at the end of the sheet.

**Example:**

In this example, the openpyxl module is utilized to load an existing Excel workbook (“sample.xlsx”). A two-dimensional data structure (tuple of tuples) is defined and iteratively appended to the active sheet, effectively adding rows with values (1, 2, 3) and (4, 5, 6).

*# import openpyxl module*

**import** **openpyxl**

wb = openpyxl.load\_workbook("sample.xlsx")

sheet = wb.active

data = (

(1, 2, 3),

(4, 5, 6)

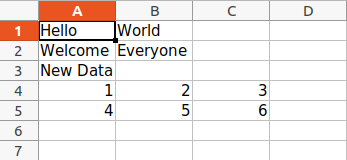
)

**for** row **in** data:

sheet.append(row)

wb.save('sample.xlsx')

**Output:**



**Arithmetic Operation on Spreadsheet**

Arithmetic operations can be performed by typing the formula in a particular cell of the spreadsheet. For example, if we want to find the sum then**=Sum()**formula of the excel file is used.

**Example:**

In this example, the openpyxl module is used to create a new Excel workbook and populate cells A1 to A5 with numeric values. Cell A7 is assigned a formula to calculate the sum of the values in A1 to A5.

*# import openpyxl module*

**import** **openpyxl**

wb = openpyxl.Workbook()

sheet = wb.active

*# writing to the cell of an excel sheet*

sheet['A1'] = 200

sheet['A2'] = 300

sheet['A3'] = 400

sheet['A4'] = 500

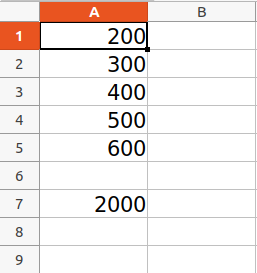
sheet['A5'] = 600

sheet['A7'] = '= SUM(A1:A5)'

*# save the file*

wb.save("sum.xlsx")

**Output:**



Refer to the below article to get detailed information about the Arithmetic operations on Spreadsheet.

* [Arithmetic operations in excel file using openpyxl](https://www.geeksforgeeks.org/python-arithmetic-operations-in-excel-file-using-openpyxl/)

**Adjusting Rows and Column**

Worksheet objects have row\_dimensions and column\_dimensions attributes that control row heights and column widths. A sheet’s row\_dimensions and column\_dimensions are dictionary-like values; row\_dimensions contains RowDimension objects and column\_dimensions contains ColumnDimension objects. In row\_dimensions, one can access one of the objects using the number of the row (in this case, 1 or 2). In column\_dimensions, one can access one of the objects using the letter of the column (in this case, A or B).

**Example:**

In this example, the openpyxl module is used to create a new Excel workbook and set values in specific cells. The content “hello” is placed in cell A1, and “everyone” is placed in cell B2. Additionally, the height of the first row is set to 70 units, and the width of column B is set to 20 units.

*# import openpyxl module*

**import** **openpyxl**

wb = openpyxl.Workbook()

sheet = wb.active

*# writing to the specified cell*

sheet.cell(row=1, column=1).value = ' hello '

sheet.cell(row=2, column=2).value = ' everyone '

*# set the height of the row*

sheet.row\_dimensions[1].height = 70

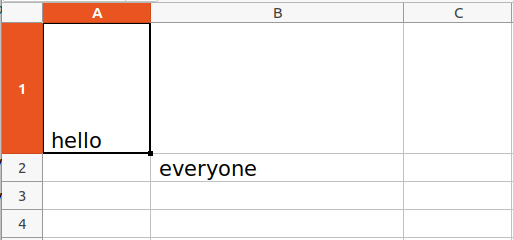
*# set the width of the column*

sheet.column\_dimensions['B'].width = 20

*# save the file*

wb.save('sample.xlsx')

**Output:**



**Merging Cells**

A rectangular area of cells can be merged into a single cell with the merge\_cells() sheet method. The argument to merge\_cells() is a single string of the top-left and bottom-right cells of the rectangular area to be merged.

**Example:**

In this example, the openpyxl module is employed to create a new Excel workbook. The program merges cells A2 to D4, creating a single cell spanning multiple columns and rows, and sets its value to ‘Twelve cells join together.’ Additionally, cells C6 and D6 are merged, and the text ‘Two merge cells.’ is placed in the resulting merged cell.

**import** **openpyxl**

wb = openpyxl.Workbook()

sheet = wb.active

sheet.merge\_cells('A2:D4')

sheet.cell(row=2, column=1).value = 'Twelve cells join together.'

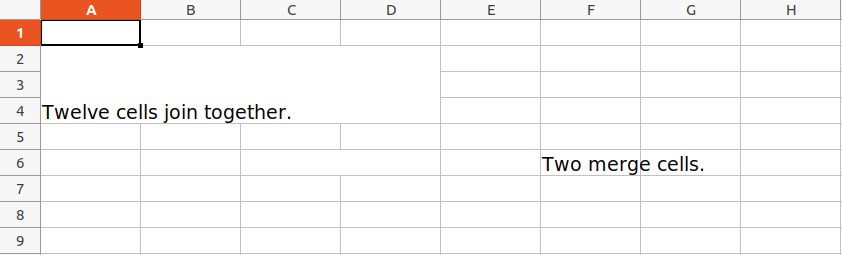
*# merge cell C6 and D6*

sheet.merge\_cells('C6:D6')

sheet.cell(row=6, column=6).value = 'Two merge cells.'

wb.save('sample.xlsx')

**Output:**



**Unmerging Cells**

To unmerge cells, call the unmerge\_cells() sheet method.

**Example:**

In this example, the openpyxl module is used to load an existing Excel workbook (“sample.xlsx”). The program then unmerges previously merged cells, specifically cells A2 to D4 and cells C6 to D6.

**import** **openpyxl**

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

sheet = wb.active

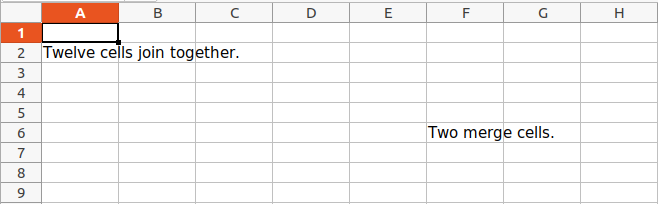
*# unmerge the cells*

sheet.unmerge\_cells('A2:D4')

sheet.unmerge\_cells('C6:D6')

wb.save('sample.xlsx')

**Output:**



**Setting Font Style**

To customize font styles in cells, important, import the **Font()** function from the **openpyxl.styles** module.

**Example:**

In this example, the openpyxl module is used to create a new Excel workbook. The program sets values in different cells with the text “GeeksforGeeks” and applies various font styles to each cell**.**

**import** **openpyxl**

*# import Font function from openpyxl*

**from** **openpyxl.styles** **import** Font

wb = openpyxl.Workbook()

sheet = wb.active

sheet.cell(row = 1, column = 1).value = "GeeksforGeeks"

*# set the size of the cell to 24*

sheet.cell(row = 1, column = 1).font = Font(size = 24 )

sheet.cell(row = 2, column = 2).value = "GeeksforGeeks"

*# set the font style to italic*

sheet.cell(row = 2, column = 2).font = Font(size = 24, italic = **True**)

sheet.cell(row = 3, column = 3).value = "GeeksforGeeks"

*# set the font style to bold*

sheet.cell(row = 3, column = 3).font = Font(size = 24, bold = **True**)

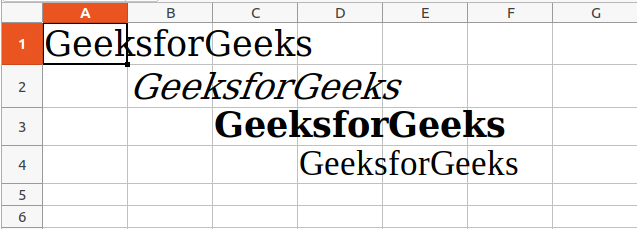
sheet.cell(row = 4, column = 4).value = "GeeksforGeeks"

*# set the font name to 'Times New Roman'*

sheet.cell(row = 4, column = 4).font = Font(size = 24, name = 'Times New Roman')

wb.save('sample.xlsx')

**Output:**



Refer to the below article to get detailed information about adjusting rows and columns.

* [Adjusting rows and columns of an excel file using openpyxl module](https://www.geeksforgeeks.org/python-adjusting-rows-and-columns-of-an-excel-file-using-openpyxl-module/)

**Plotting Charts**

Charts are composed of at least one series of one or more data points. Series themselves are comprised of references to cell ranges. For plotting the charts on an excel sheet, firstly, create chart objects of specific chart class( i.e BarChart, LineChart, etc.). After creating chart objects, insert data in it, and lastly, add that chart object in the sheet object.

**Example 1: Creating and Customizing Bar Chart in Excel with openpyxl**

In this example, the openpyxl module is used to create a new Excel workbook. Numeric values from 0 to 9 are written to the first column of the active sheet. A BarChart object is then created, and data for plotting is specified using the Reference class. The chart is customized with a title, x-axis title, and y-axis title. Finally, the chart is added to the sheet, anchored to cell E2.

*# import openpyxl module*

**import** **openpyxl**

*# import BarChart class from openpyxl.chart sub\_module*

**from** **openpyxl.chart** **import** BarChart, Reference

wb = openpyxl.Workbook()

sheet = wb.active

*# write o to 9 in 1st column of the active sheet*

**for** i **in** range(10):

sheet.append([i])

*# create data for plotting*

values = Reference(sheet, min\_col=1, min\_row=1,

max\_col=1, max\_row=10)

*# Create object of BarChart class*

chart = BarChart()

*# adding data to the Bar chart object*

chart.add\_data(values)

*# set the title of the chart*

chart.title = " BAR-CHART "

*# set the title of the x-axis*

chart.x\_axis.title = " X\_AXIS "

*# set the title of the y-axis*

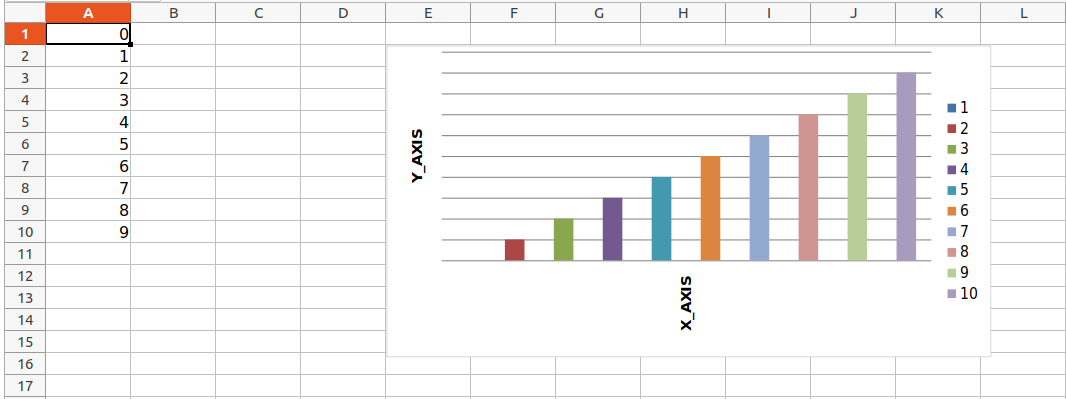
chart.y\_axis.title = " Y\_AXIS "

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

*# save the file*

wb.save("sample.xlsx")

**Output:**



**Example 2: Creating and Customizing Line Chart in Excel with openpyxl**

In this example, the openpyxl module is used to create a new Excel workbook. Numeric values from 0 to 9 are written to the first column of the active sheet. A LineChart object is then created, and data for plotting is specified using the Reference class. The chart is customized with a title, x-axis title, and y-axis title. Finally, the chart is added to the sheet, anchored to cell E2.

*# import openpyxl module*

**import** **openpyxl**

*# import LineChart class from openpyxl.chart sub\_module*

**from** **openpyxl.chart** **import** LineChart, Reference

wb = openpyxl.Workbook()

sheet = wb.active

*# write o to 9 in 1st column of the active sheet*

**for** i **in** range(10):

sheet.append([i])

values = Reference(sheet, min\_col=1, min\_row=1,

max\_col=1, max\_row=10)

*# Create object of LineChart class*

chart = LineChart()

chart.add\_data(values)

*# set the title of the chart*

chart.title = " LINE-CHART "

*# set the title of the x-axis*

chart.x\_axis.title = " X-AXIS "

*# set the title of the y-axis*

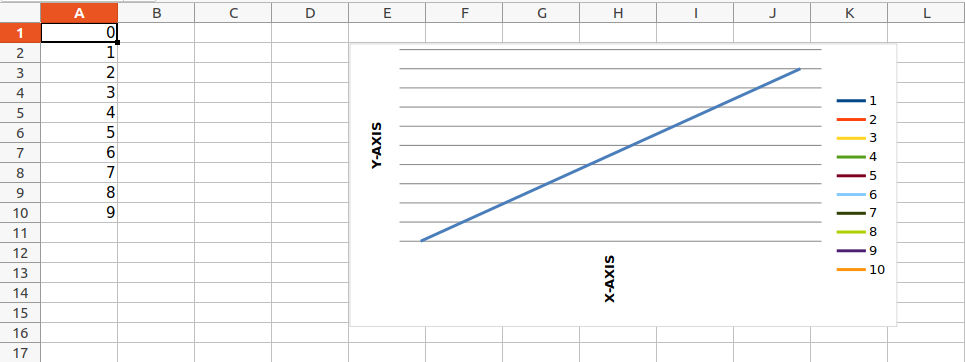
chart.y\_axis.title = " Y-AXIS "

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

*# save the file*

wb.save("sample.xlsx")

**Output:**



Refer to the below articles to get detailed information about plotting in excel using Python.

* [Plotting charts in excel sheet using openpyxl module | Set  1](https://www.geeksforgeeks.org/python-plotting-charts-in-excel-sheet-using-openpyxl-module-set-1/)
* [Plotting charts in excel sheet using openpyxl module | Set  2](https://www.geeksforgeeks.org/python-plotting-charts-in-excel-sheet-using-openpyxl-module-set-2/)
* [Plotting charts in excel sheet using openpyxl module | Set 3](https://www.geeksforgeeks.org/python-plotting-charts-in-excel-sheet-using-openpyxl-module-set-3/)

**Adding Images**

For the purpose of importing images inside our worksheet, we would be using **openpyxl.drawing.image.Image.** The method is a wrapper over PIL.Image method found in PIL (pillow) library. Due to which it is necessary for the PIL (pillow) library to be installed in order to use this method.

**Image Used:**



**Example:**

In this example, the openpyxl module is utilized to create a new Excel workbook. A row of data is added to the active sheet to distinguish it from the image. An image (“geek.jpg”) is then added to the worksheet using the openpyxl.drawing.image.Image class, and it is positioned at cell A2.

**import** **openpyxl**

**from** **openpyxl.drawing.image** **import** Image

wb = openpyxl.Workbook()

sheet = wb.active

sheet.append([10, 2010, "Geeks", 4, "life"])

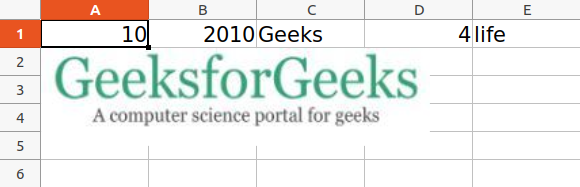
img = Image("geek.jpg")

sheet.add\_image(img, 'A2')

*# Saving the workbook created*

wb.save('sample.xlsx')

**Output:**



Refer to the below article to get detailed information about adding images.

* [Openpyxl – Adding Image](https://www.geeksforgeeks.org/openpyxl-adding-image/)

**Some More Functionality of Excel using Python**

* [How to delete one or more rows in excel using Openpyxl?](https://www.geeksforgeeks.org/how-to-delete-one-or-more-rows-in-excel-using-openpyxl/)
* [Trigonometric operations in excel file using openpyxl](https://www.geeksforgeeks.org/working-with-excel-spreadsheets-in-python/geeksforgeeks.org/python-trigonometric-operations-in-excel-file-using-openpyxl/)
* [How to copy data from one excel sheet to another](https://www.geeksforgeeks.org/python-how-to-copy-data-from-one-excel-sheet-to-another/)
* [How to Automate an Excel Sheet in Python?](https://www.geeksforgeeks.org/how-to-automate-an-excel-sheet-in-python/)

**Working with Excel Spreadsheets in Python – FAQs**

**How to Use Excel with Python?**

*To use Excel with Python, you can utilize libraries like pandas, openpyxl, or xlrd/xlwt. These libraries help read, write, and manipulate Excel files. pandas is particularly useful for data analysis and manipulation and can easily read and write Excel files using the read\_excel() and to\_excel() functions.*

***Example with pandas:***

*import pandas as pd  
  
# Reading an Excel file  
df = pd.read\_excel('filename.xlsx')  
  
# Manipulating data  
df['new\_column'] = df['existing\_column'] \* 10  
  
# Writing to an Excel file  
df.to\_excel('modified\_filename.xlsx', index=False)*

**Can I Automate Excel with Python?**

*Yes, you can automate Excel tasks with Python using libraries such as openpyxl for handling .xlsx files or xlwings to interact with Excel applications. These tools allow you to automate repetitive tasks like formatting cells, inserting data, and creating charts.*

***Example with openpyxl:***

*from openpyxl import Workbook  
  
wb = Workbook()  
ws = wb.active  
  
# Adding data  
ws['A1'] = "Hello"  
ws['A2'] = "World!"  
  
# Save the workbook  
wb.save("example.xlsx")*

**What is the Difference Between pandas and openpyxl?**

* ***pandas*** *is a high-level data manipulation tool designed primarily for data analysis. It provides functionality for reading and writing to Excel files but is not capable of interacting with Excel files at a low level (like modifying cell styles or other formatting features).*
* ***openpyxl*** *is a library dedicated to reading from and writing to Excel .xlsx files. It allows more granular control over Excel files, such as adjusting cell styles, filters, and formulas, which pandas does not directly allow.*

**Can We Import Excel File in Python?**

*Yes, you can import Excel files in Python using several libraries. pandas is the most common library for dealing with structured data files, including Excel, because it can quickly load an Excel file into a DataFrame, allowing for extensive data manipulation and analysis.*

***Example with pandas:***

*import pandas as pd  
  
# Load an Excel file into a DataFrame  
df = pd.read\_excel('your\_file.xlsx')  
print(df)*

**Can You Use Python in Excel Instead of VBA?**

*Yes, Python can be used in Excel as an alternative to VBA through libraries like xlwings. This library allows Python code to interact directly with Excel, leveraging Python’s syntax and capabilities for automation, data analysis, and visualization within Excel. This is particularly useful for users looking to utilize the advanced capabilities of Python without leaving the Excel environment.*

***Example with xlwings:***

*import xlwings as xw  
  
# Connect to an existing Excel workbook  
wb = xw.Book('example.xlsx')  
sheet = wb.sheets['Sheet1']  
  
# Write data to Excel  
sheet.range('A1').value = 'Hello, Excel!'  
  
# Run Excel formulas  
sheet.range('A2').value = '=SUM(1, 2, 3)'  
  
wb.save()  
wb.close()*

*These tools and methods make Python a powerful companion for working with Excel, enabling both simple data manipulations and complex automations*