2020

Free Python Course

O. Elijah
onlinepythonclass@gmail.com

O. ELIJAH (MSc, Ph.D)

1 WORKING WITH FILES AND DATA VISUALIZATION

In this chapter, we will see how to work with files and visualize data in excel. This will include creating a new file, opening a file, closing the file, reading a file, writing to the file, manipulating the file and finally creating a copy of the file. So, sit down and relax like a programmer with a glass of water. We shall work with text file and excel file.

In Windows, for example, a file can be any item manipulated, edited or created by the user/OS. That means files can be images, text documents, executables, and much more. Most files are organized by keeping them in individual folders.

In Python, a file is categorized as either text or binary, and this difference is important.

Hence, in this chapter you will be dealing with text and bytes of information.

1.1 Review of Chapter 6

In chapter six we looked at how to create a function and how to work with modules. In this chapter we shall use modules such as openpyxl for excel application.

1.2 New Folder

Create a new folder on your desktop and name it **WEEK 7**. Download the text file named **testfile,** the image file named **success**, and the excel file named **my_data** from the LMS and save them in the **WEEK 7** folder.

Create a new python file using Spyder or any other python IDE you are using and save it in the **WEEK 7** folder. This is to ensure all the files are in the same folder. If your python file is different from the other files you will need to provide the full path to the file in your python code.

1.3 Working with text file

We shall explore how to open, read, add and delete an existing text file.

1.4 The Open Function

The open function simple open a document and assign it to a file object or a handler. The function can be in two forms

```
ourfile=open('filename.extension')
```

or

```
ourfile1=open('filename.extension','mode')
```

the first form will open the file with a default mode (read) while the second form will open the file in the specified mode. It is always good to use the second mode except you only want to read the file.

Let's try out some open functions.

Open the testfile in your folder and print out the handler.

```
ourfile=open('testfile.txt','r')
print(ourfile)
```

OUT:

```
<_io.TextIOWrapper name='testfile.txt' mode='r' encoding='cp1252'>
```

the output tells you the details about the file such as the name, the mode, and the encoding.

The details about the available modes will be discussed shortly. We could view only the name by

```
ourfile=open('testfile.txt','r')
print(ourfile.name)
```

or mode by

```
ourfile=open('testfile.txt','r')
print(ourfile.mode)
```

It is also important to explicitly close the file after you finish with it, else it could begin to cause an error when you run over the maximum allowed file descriptors on your system. To close the file simple use the handler.close().

In our example we have

```
ourfile=open('testfile.txt','r')
```

```
print(ourfile.mode)
ourfile.close()
```

Now before we continue let's go over the different modes available to us.

1.5 Modes

1	**R** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**Rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**W** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6	**Wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**wb+**

	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	**A** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	**Ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Note that the modes are all in small case letters. We will be using some of these modes and you can try a lot more on your own.

Now recall that you have to explicitly close the handler after you finish using it. Another way to open a file is using a context manager.

1.6 Open File Using A Context Manager

This is similar to the open method we did before, except that the open is executed within a context and the hander automatically close the file when you exit the context or if any exceptions are thrown. For examples.

```
with open('testfile.txt','r') as ourfile:
    pass
print(ourfile.closed)
```

Notice that the print statement is outside the context manager and it can confirm with a TRUE result to inform you that the file is closed outside the context (handle outside the context). However, because the file is closed you won't be able to read from the file again which means you can't manipulate it outside the context. From henceforth, we will be working with the file inside the context manager (best practises).

1.7 Reading the content of file

Now we have been able to open out file and we understand the different modes available. How do we get the content of our file?

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.read()
    print(content)
```

Out:

1. This is the sixth chapter of free python class
2. I can't tell you how much I enjoy it so far
3. So what have you learnt so far?
4. A lot I just can't tell you now
5. But I see that I am becoming a programmer
6. A python Programmer are you kidding me?
7. Yes! it's been an interesting class
8. I am excitedly waiting for the project
9. I will be able to put everything to practice soon
10. Could you teach me when you are done learning?

Good job!!!. Well this method of read is good if you have a small file. However, we may have a large file and don't want to load all the content of that file into memory at once. There are couple of ways to do that.
We could use readline or readlines

Example

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.readline()
    print(content)
```

Out:

```
1. This is the sixth chapter of free python class
```

This reads only one line. We can simply print out the next line by reading it again

```
with open('testfile.txt','r') as ourfile:
```

```
    content=ourfile.readline()
    print(content)
    content=ourfile.readline()
    print(content)
```

```
output

1. This is the sixth chapter of free python class

2. I can't tell you how much I enjoy it so far
```

Notice that the list has a new gap that was not in the original testfile, this is because the print statement by default ends with a new line we can fix that by passing in an empty string to the end of the print. Please check out the output

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.readline()
    print(content, end='')
    content=ourfile.readline()
    print(content, end='')
    content=ourfile.readline()
    print(content, end='')
```

you could also use the readlines command

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.readlines()
    print(content, end='')
```

```
Out:

['1. This is the sixth chapter of free python class\n', "2. I can't
tell you how much I enjoy it so far\n", '3. So what have you learnt so
far?\n', "4. A lot I just can't tell you now\n", '5. But I see that I
am becoming a programmer\n', '6. A python Programmer are you kidding
me?\n', "7. Yes! it's been an interesting class\n", '8. I am excitedly
waiting for the project\n', '9. I will be able to put everything to
practice soon\n', '10. Could you teach me when you are done learning?']
```

Notice it read all the line with the \n character to inform you of a new line. However, for a large file we may not want to use the readlines or the read line command. This is because reading the who file at once may use up the system memory, and reading line after line maybe be time consuming.

One solution is by using a for loop to print line by line

Example

```
with open('testfile.txt','r') as ourfile:
     for line in ourfile:
        print(line, end='')
```

This is an efficient way because it doesn't read all the content to memory at once, it simple read each line to memory.

Another way to handle large file is by using the read command but this time pass in an argument to read a chunk at a time

Example

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.read(100)
    print(content, end='')
```

this reads the first 100 character

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.read(100)
    print(content, end='')

    content=ourfile.read(100)
    print(content, end='')
```

this reads the first two hundred characters by reading only a 100 character to memory at a time, if we repeat the read and print command long enough to the end of our file, the print command will simply return an empty string. But since we don't know how large the file can be, we could just do this in a loop and set the condition to check for the empty string.

Example

```
with open('testfile.txt','r') as ourfile:
    chunk_to_read=100
    content=ourfile.read(chunk_to_read)

    while len(content) > 0:
        print(content, end='')
        content=ourfile.read(chunk_to_read)
```

This returns the same result as before and print out the entire testfile, only that is does that by loading 100 characters to memory each time. To see this effect, you could try to change the print statement by adding an extra character after each print.

Hint: change the line `print(content, end='')` to `print(content, end='*')` and watch out for the asterisk which appears after every 100 character

1.7.1 Tell command

Note that each time we print the content in the file, you can observe that the counter advances it positions that is why it is able to print the next 100 characters. You can check out the current position by using the "`handler.tell()`" command

Example

```
with open('testfile.txt','r') as ourfile:
```

```
    chunk_to_read=100
    content=ourfile.read(chunk_to_read)
    print(ourfile.tell())
```
this tells you the current position of the pointer in the file.

1.7.2 Seek command

The seek command can take the pointer to a new position other than advancing the pointer. The seek command can be expressed as "`handler.seek()`" for example.

```
with open('testfile.txt','r') as ourfile:
    content=ourfile.read(10)
    print(content, end='')

    content=ourfile.read(10)
    print(content, end='')

    ourfile.seek(0)
    content=ourfile.read(10)
    print(content, end='')
```

Notice that after printing the first 20 characters the seek position forces the pointer back to position 0 and the next print starts from the beginning. You can try out new positions.

Okay nice Job to this point lets learn some new things like writing to file.

1.8 Writing to file

Now I hope you remember the discussion on modes (see section 6.4). You cannot write to a file if the file is open in read only mode. The write command is similar to read command and by simple using the "`handler.write()`" you can write to a file.

Important thing to note is that if the file does not exist the write mode will create the file. But if it does already exist the file will be replaced. So, what if I don't want to override the file, I just want to add more content to it? You only need to change the mode to lower case "a" which mean you want to append to the file. More details can be found in table 6.1.

Now let create a new file and write the title of this chapter to the file

```
with open('newfile.txt','w') as ourfile:
    ourfile.write('Working with Files')
```

You can check your folder and you will see the new file and the content will just be what you wanted. You could try to write additional stuff there. Hint: replicate the write line `ourfile.write('Working with Files')` and change its content. You can also use the \n to print in a new line

You can also use the seek discussed in 6.6.2 to change the pointer to where you want to write next.

Example

```
with open('newfile.txt','w') as ourfile:
    ourfile.write('Working with Files')
    ourfile.seek(0)
    ourfile.write('playing')
```

Okay now that you understand read and write let's put it all together.

And for this example, we will read from our "testfile" and write (copy) it's content into a new file "testcopy". To read please refer to section 6.6. you could use any of the methods

```
with open('testfile.txt','r') as our_read_file:
    with open('testcopy.txt','w') as our_write_file:
        for line in our_read_file:
            our_write_file.write(line)
```

Feeling good right? You could check the folder to see the testcopy you just made, and the content will be exactly what is inside the original file. Come let's try to create a copy of a large image file. There is an image file named **success** and the format is jpeg in your WEEK 7 folder.

Lets try this.

```
with open('pexels.jpeg','r') as our_read_file:
    with open('pexelscopy.jpeg','w') as our_write_file:
        for line in our_read_file:
            our_write_file.write(line)
```

I did everything correctly, even the file format, yet I got an error WHY??

Yes, it throws and error because it is not a text document therefore you cannot use read and write ('r', 'w'). You have to read and write the file in binay mode therefore you have to use read byte and write byte ('rb', 'wb')

example

```
with open('success.jpg','rb') as our_read_file:
    with open('pexelscopy.jpeg','wb') as our_write_file:
        for line in our_read_file:
            our_write_file.write(line)
```

Result

A copy of the image has been saved in your WEEK 7 folder as **pexelscopy** in jpeg format.

Like in the previous example, we can also choose to read a chunk of data at a time and write the chunk to the new file.

Example

```
with open('success.jpg','rb') as our_read_file:
      with open('pexelscopy.jpeg','wb') as our_write_file:
          chunk_size=4096 #4bytes(1byte=1024bits)
          rf_chunck=our_read_file.read(chunk_size)
          while len(rf_chunck)>0:
              our_write_file.write(rf_chunck)
              rf_chunck=our_read_file.read(chunk_size)
```

1.9 Adding to an existing file

You could also want to manipulate and existing text by deleting or writing to the existing file. To achieve this, you need to change the mode to append.

```
From our previous example
with open('testfile.txt','a+') as ourfile:
    ourfile.write('\n11. This is the new line from chapter 6')
```

The + sign in a+ indicates that the file should be created if it does not exist however, in our case it exist already so you should see this (`11. This is the new line from chapter 6`) added to you file.

1.10 Deleting lines from an existing file

You could also delete line from an existing text by removing the line.

Example

```
with open('testfile.txt','r') as our_read_file:
    lines = our_read_file.readlines()
    lines.remove("11. This is the new line from chapter 6\n")
    with open('testcopy.txt','w') as our_write_file:
        for line in lines:
            our_write_file.write(line)
```

```
The remove function can also be used to remove a file from your systems.
However, you need to write out the path to the file except if the file is in
your current working directory. To simple remove a file from the current
directory you need to import os then remove the file by
os.remove("filename.extension")
```

Example

```
import os
os.remove("newfile.txt")
```

1.10.1 Renaming file

You could rename a file using the following example

```
import os

os.rename( "currentname.extension", "newname.extension" )
```

1.11 Woking with excel files and visualizing data in excel file

Now let us examine how to work with excel file. Note there are different python modules suitable for excel. Examples are the openpyxl, pandas, xlwt and xlrd. We shall work with the openpyxl.

open an excel file my_data in the **WEEK 7** folder, use the following commands

Important things to note, the make sure the excel file you are working with is not opened on your computer while writing your python codes to it. If open it will display an error message.

First you need to import the **openpyxl** module and load the **my_data** excel file into python as and object

```
import openpyxl
wb_obj = openpyxl.load_workbook('my_data.xlsx') #  Open excel work book
# Get workbook active sheet object
# from the active attribute
sheet_obj = wb_obj.active
```

Read a value from the excel first column and first row in my_data.xlsx

```
# Note: The first row or
# column integer is 1, not 0.
# Cell object is created by using
# sheet object's cell() method.
cell_obj = sheet_obj.cell(row = 1, column = 1)
print(cell_obj.value)
```

Result

```
Name
```

Change the row to 1 and column to 4 and observe the values from the excel sheet

Result

```
Salary
```

Determine the total number of rows and columns

```
# determine the total number of columns
sheet_obj = wb_obj.active

# ptint total number of column
print("total number of column is: ", sheet_obj.max_column)

# determine the total number of Rows
print("\ntotal number of row is: ", sheet_obj.max_row)
```

Result

```
total number of column is:  4

total number of row is:  12
```

Read all column names from the excel

```
# print all column names
max_col = sheet_obj.max_column
# Loop will print all columns name
for i in range(1, max_col + 1):
    cell_obj = sheet_obj.cell(row = 1, column = i)
    print(cell_obj.value)
```

Result

```
Name

Age

Gender

Salary
```

Read first column value. Note the index in starts from 1 and not 0 when reading from excel.

```
# first column value
m_row = sheet_obj.max_row
# Loop will print all values
# of first column
print("\nfirst column values")
```

```
for i in range(1, m_row + 1):
    cell_obj = sheet_obj.cell(row = i, column = 1)
    print(cell_obj.value)
```

Result

```
first column values

Name

john

Mary

Ali

Lisa

abdul

Jen

musa

Azmil

Kong

Fatin
```

Read a particular row

```
# Print a particular row value
max_col = sheet_obj.max_column
# Will print a particular row value
print("All values in row 2")
for i in range(1, max_col + 1):
    cell_obj = sheet_obj.cell(row = 2, column = i)
    print(cell_obj.value, end = " ")
```

Result

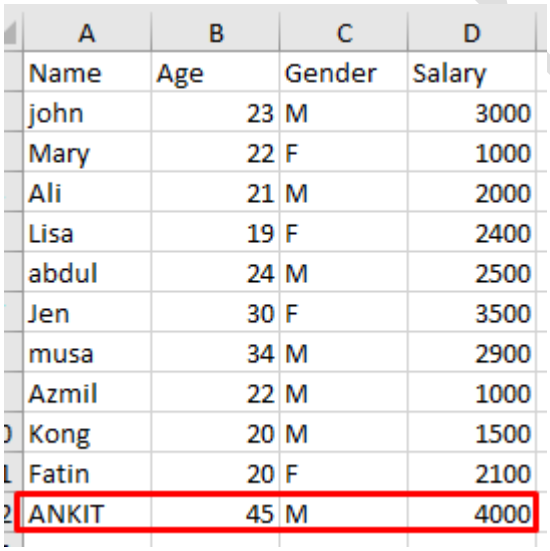
```
All values in row 2

john  23 M 3000
```

Write to excel. Let us add new data to the excel sheet Name is ANKIT, age is 45, Gender is M and Salary is 4000.

```
#writing to excel sheet1

# Note: The first row or column integer
# is 1, not 0. Cell object is created by
# using sheet object's cell() method.
c1 = sheet_obj.cell(row = 12, column = 1)
c2 = sheet_obj.cell(row = 12, column = 2)
c3 = sheet_obj.cell(row = 12, column = 3)
c4 = sheet_obj.cell(row = 12, column = 4)

# writing values to cells
c1.value = "ANKIT"
c2.value = 45
c3.value = "M"
c4.value = 4000

# make sure your my_data excel workbook is closed
# save the data to excel
wb_obj.save('my_data.xlsx')
```

Result

Open your my_data excel using Microsoft office. You will see the new data



	A	B	C	D
	Name	Age	Gender	Salary
	john	23	M	3000
	Mary	22	F	1000
	Ali	21	M	2000
	Lisa	19	F	2400
	abdul	24	M	2500
	Jen	30	F	3500
	musa	34	M	2900
	Azmil	22	M	1000
	Kong	20	M	1500
	Fatin	20	F	2100
	ANKIT	45	M	4000

Visualize date in excel using barchart.

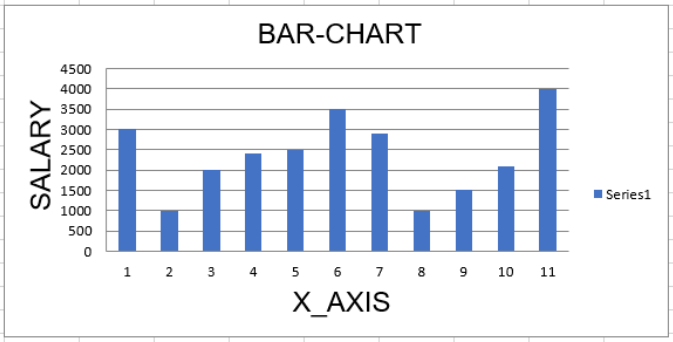
Let us plot the Salary data in column 4, from row 2 to row 12 and save it to a new excel file titled **my_data_barchart.xlsx**

```
# import openpyxl module
import openpyxl

# import BarChart class from openpyxl.chart sub_module
from openpyxl.chart import BarChart, LineChart, Reference

# workbook object is created
wb_obj = openpyxl.load_workbook('my_data.xlsx') #  Open excel work book

# Get workbook active sheet
# from the active attribute.
sheet = wb_obj.active

#1 : Plot the Bar Chart

# create data for plotting
values = Reference(sheet, min_col = 4, min_row = 2,
                          max_col = 4, max_row = 12)

# 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 = " SALARY "

# add chart to the sheet
# the top-left corner of a chart
# is anchored to cell H2 .
sheet.add_chart(chart, "H2")

# save the file
wb_obj.save("my_data_barchart.xlsx")
```

Result

Check your current folder WEEK 7 and open the excel file named my_data_barchart.xlsx you will see the bar chart

Name	Age	Gender	Salary
john	23	M	3000
Mary	22	F	1000
Ali	21	M	2000
Lisa	19	F	2400
abdul	24	M	2500
Jen	30	F	3500
musa	34	M	2900
Azmil	22	M	1000
Kong	20	M	1500
Fatin	20	F	2100
ANKIT	45	M	4000



Visualize date in excel using Line chart.

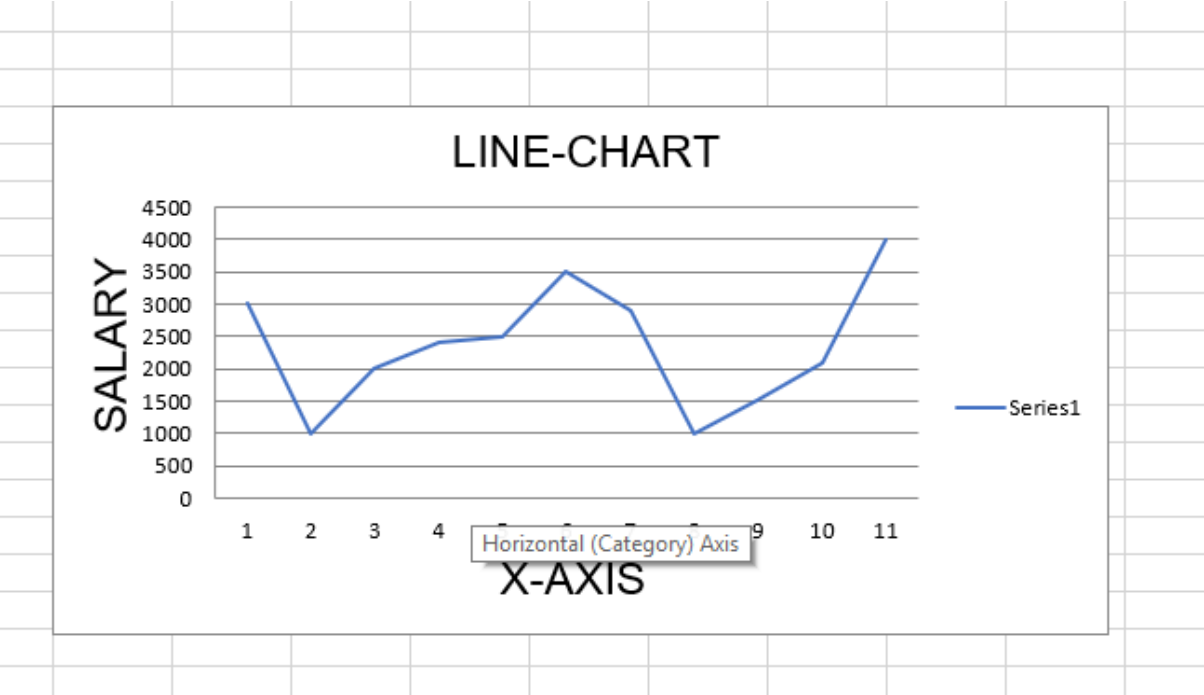
Let us plot the Salary data in column 4, from row 2 to row 12 and save it to a new excel file titled **my_data_linechart.xlsx**

```
import openpyxl module
import openpyxl

# import BarChart class from openpyxl.chart sub_module
from openpyxl.chart import BarChart, LineChart, Reference

# workbook object is created
wb_obj = openpyxl.load_workbook('my_data.xlsx') #  Open excel work book

# Get workbook active sheet
# from the active attribute.
sheet = wb_obj.active

#1 : Plot the Bar Chart

# create data for plotting
values = Reference(sheet, min_col = 4, min_row = 2,
                          max_col = 4, max_row = 12)

# 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 = " SALARY "

# add chart to the sheet
# the top-left corner of a chart
# is anchored to cell H25 .
sheet.add_chart(chart, "H25")

# save the file
```

```
wb_obj.save("my_data_linechart.xlsx")
```

Result

Check your current folder WEEK 7 and open the excel file named my_data_linechart.xlsx you will



see the bar chart

1.12 Application

Example 1.

We have a data shown the following table

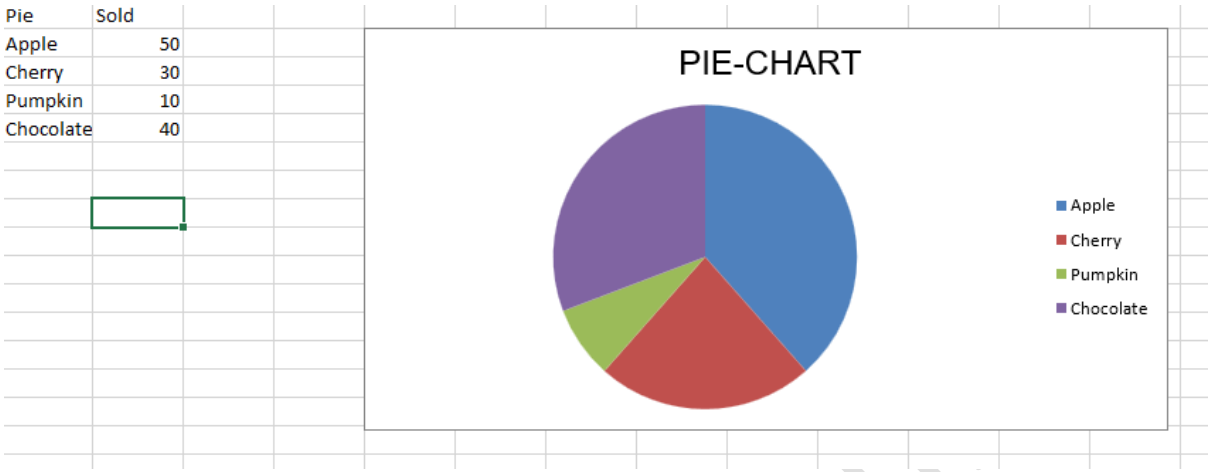
Items	Sold
Apple	50
Cherry	30
Pumpkin	10
Chocolate	40

Let's write python code to save the data into an excel sheet in our current folder WEEK 7 and plot a pie chart. To do this use the following codes

```
# import openpyxl module
import openpyxl

# import PieChart, Reference class
# from openpyxl.chart sub_module
from openpyxl.chart import PieChart, Reference

# Call a Workbook() function of openpyxl
# to create a new blank Workbook object
wb = openpyxl.Workbook()

# Get workbook active sheet
# from the active attribute.
sheet = wb.active

datas = [
    ['Pie', 'Sold'],
    ['Apple', 50],
    ['Cherry', 30],
    ['Pumpkin', 10],
    ['Chocolate', 40],
]


# write content of each row in 1st, 2nd and 3rd
# column of the active sheet respectively.
for row in datas:
    sheet.append(row)

# Create object of PieChart class
chart = PieChart()

# create data for plotting
labels = Reference(sheet, min_col = 1,
                   min_row = 2, max_row = 5)

data = Reference(sheet, min_col = 2,
                   min_row = 1, max_row = 5)

# adding data to the Pie chart object
chart.add_data(data, titles_from_data = True)

# set labels in the chart object
chart.set_categories(labels)

# set the title of the chart
chart.title = " PIE-CHART "

# add chart to the sheet
# the top-left corner of a chart
# is anchored to cell E2 .
sheet.add_chart(chart, "E2")

# save the file
wb.save(" PieChart.xlsx")
```

Result

Open your week 7 folder and you will see the new excel file named PieChart. Open the excel file to view the data and pie chart.



1.13 Reference

[1] https://www.tutorialspoint.com/python/python_files_io.htm

[2] https://www.tutorialspoint.com/python/os_file_methods.htm

[3] https://www.w3resource.com/python-exercises/file/index.php

[4]https://www.geeksforgeeks.org/python-plotting-charts-in-excel-sheet-using-openpyxl-module-set-2/

1.14 Exercise

1. Write a Python program that accept input from keyboard and append the input text to an existing file.
2. Write a Python program to read a file line by line and store is as
    a. A list
    b. A variable
    c. An array

    Hint: https://www.w3resource.com/python-exercises/file/index.php

3. Write a Python program to write the following data to an excel file named student_record

   The following data should be saved into the file.

Name	ID	Contact number	Age	Scores
John	01	08034040412	17	78
Musa	02	07034040432	18	80
Fatin	03	07094034982	16	85
Linda	04	08909345213	17	90

   Plot a line graph of the ages of the students and plot a bar chart of the scores of the students in the same excel sheet.

Exercise is due on the 4Th of March