Python Lab1: Excel with openpyxl

Visual Programming

R. L. Zimmerman

Table of Contents

1. Introduction

In this chapter we discuss

The Lab
Python
What's Next?

Are you curious about the Python language and wondering how to read and write Excel files? This manual uses the format of a hands-on Lab, with simple code examples that perform one basic task: compare two Excel files and output differences to a third Excel file. At the end of the Lab, you will know enough about Python to work with your own Excel files, even if you're new to Python or programming.

This manual uses a Python script file. A script is a simple text file with the ".py" extension. I tried to write the manual in such a way that beginners or experienced programmers could quickly find what they're looking for, skipping over details they're not interested in right now. The content is laid out and indexed so you can always come back and review those topics later.

The Lab of step-by-step examples walks through each line of code. There are over 40 illustrations. Since Project1 is only 50 lines of code, I may have gone overboard on the illustrations. I wanted you to visually

see the code in action with numbered examples, screenshots, and tables. The written explanation and graphics highlight the line numbers in the code, so you can follow along and visualize the code running.

Download the sample Excel files and code from my Github account, as shown in the Appendix. The download includes this manual in PDF form; I'd encourage you to print the PDF and keep it by your side as you work through the Lab. While I can't be there in person to answer your questions, the printed PDF is the next best thing. Please feel free to copy and share the materials with others. If you'd be kind enough to leave a review, it might lead to another $1 royalty in my future!

1.1 The Lab

The Lab has two parts. Part 1 accomplishes the basic tasks to compare the two Excel files and create the third Excel file. I think of this as the core program that gets the job done. Part 2 adds some nice-to-have features and shows how to customize the code for use with your own Excel files.

Hopefully, a working code example will take all the guesswork out of programming, leaving just the fun of learning something new. With this Lab, you don't have to wonder if you have the correct indentation, your counter is in the right place, or if you forgot the colon at the end of the line when you defined your function.

1.2 Python

Python is an open-source (free) programming language for Web Development, GUI development, Scientific and Numeric data science, Software Development, and System Administration. This Lab uses the open-source Anaconda Data Science Distribution that includes Python version 3.7. Spyder, the Scientific Python Development Environment, comes with Anaconda, and we'll write and run a Python script in Spyder on a Windows machine.

1.3 What's Next?

The next Chapter walks you through installing Anaconda and the basics of running code in Spyder.

2. Setup & Getting Started

In this Chapter we discuss

Anaconda
Lab Files
Spyder
Run Project1.1.py File
What's Next?

In this Chapter we'll install Anaconda, set up your environment, and run the program Project1.1.

2.1 Anaconda

Download the Anaconda Distribution for Windows that includes Python version 3.7. Other versions will work but may vary slightly

compared to the examples in this manual. When prompted, update your Path settings. The install will take a while, so you might want to grab a cup of coffee or something.

2.2 Lab Files

Download the files for this Lab and place them in the same folder. See the Appendix for details on the download Github location. It doesn't matter which folder you use because Spyder prompts you for this working directory when you run your program. The next figure displays my working directory and sample project files.

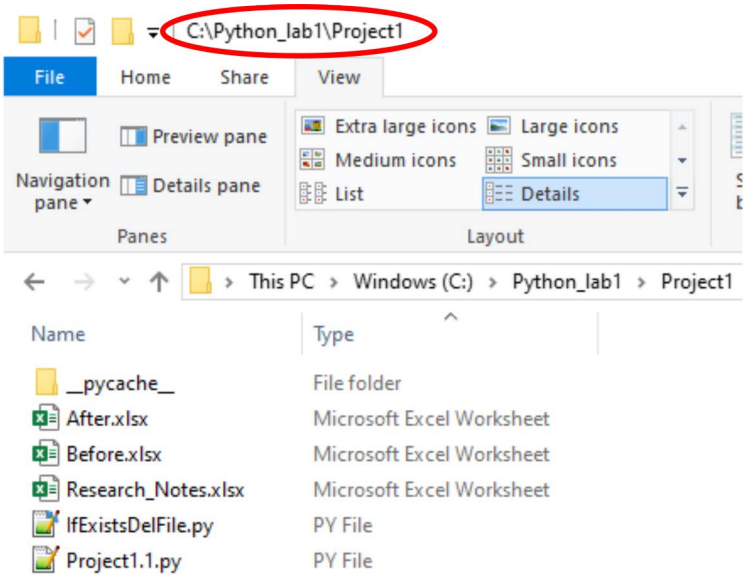


Figure 2.1 Project Files

2.3 Spyder

Launch Spyder from the Start Menu, in the Anaconda folder.

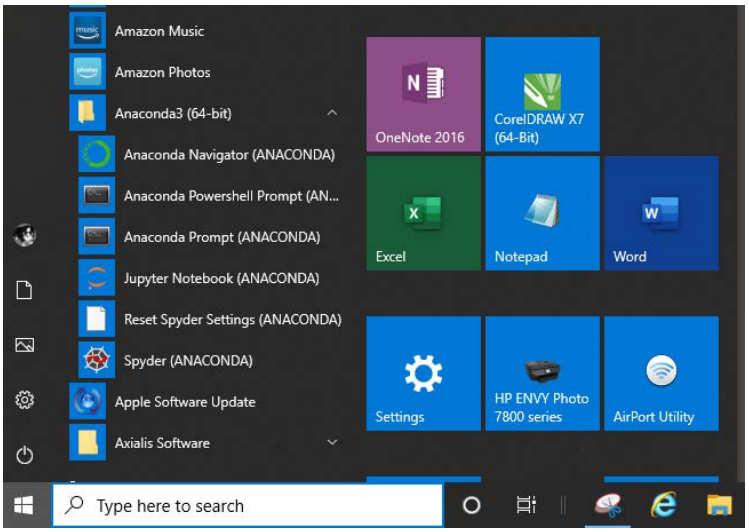


Figure 2.2 Launch Spyder

The Spyder Default Layout has three panes, as shown below. You can return to this layout at any time from the View menu under Windows Layouts. You can close or open other panes to suit your preferences.

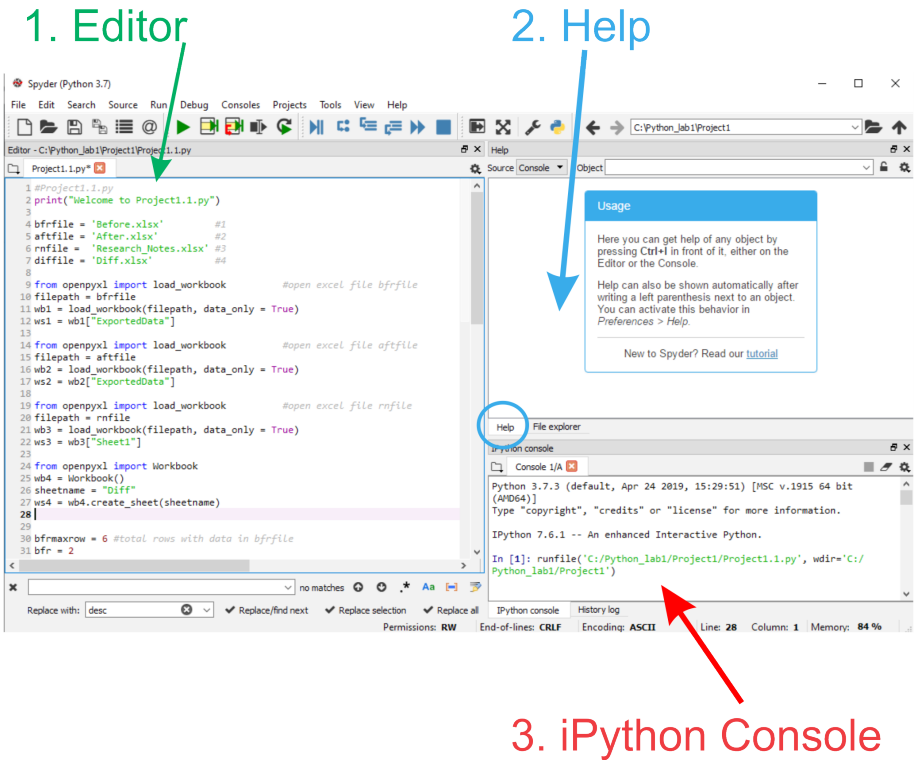


Figure 2.3 Project Files

1. The Editor window is where you type your code.

2. The Help window displays syntax and function help.

3. The iPython Console. When you click "Run," the results are output to the Console. Results include code output and error messages. For example, if you use the Print method, the results are output to the Console window. In the example below, the Console displays "**Welcome to Project1.1.py**."

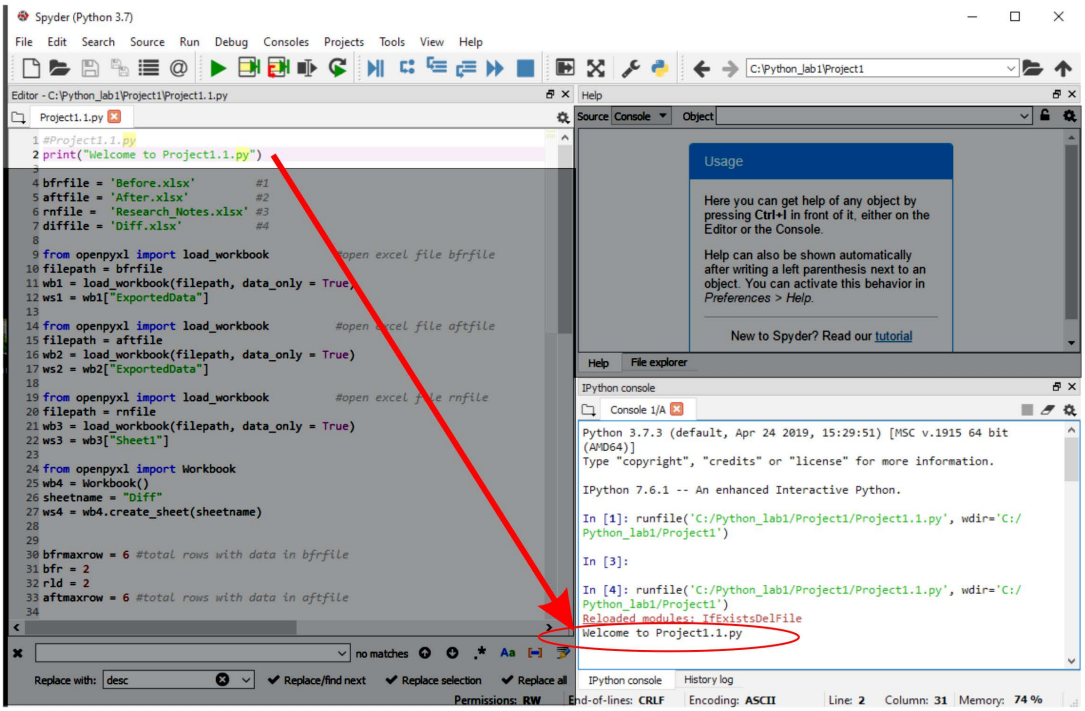


Figure 2.4 The iPyhton Console

2.4 Run Project1.1.py File

With Spyder open, click on the File menu, and then click on "Open" and browse to the directory with the Lab files. Open the "*Project11..py*" file. The Console window To run the code, click on the green arrow or use the Run menu, as shown below.

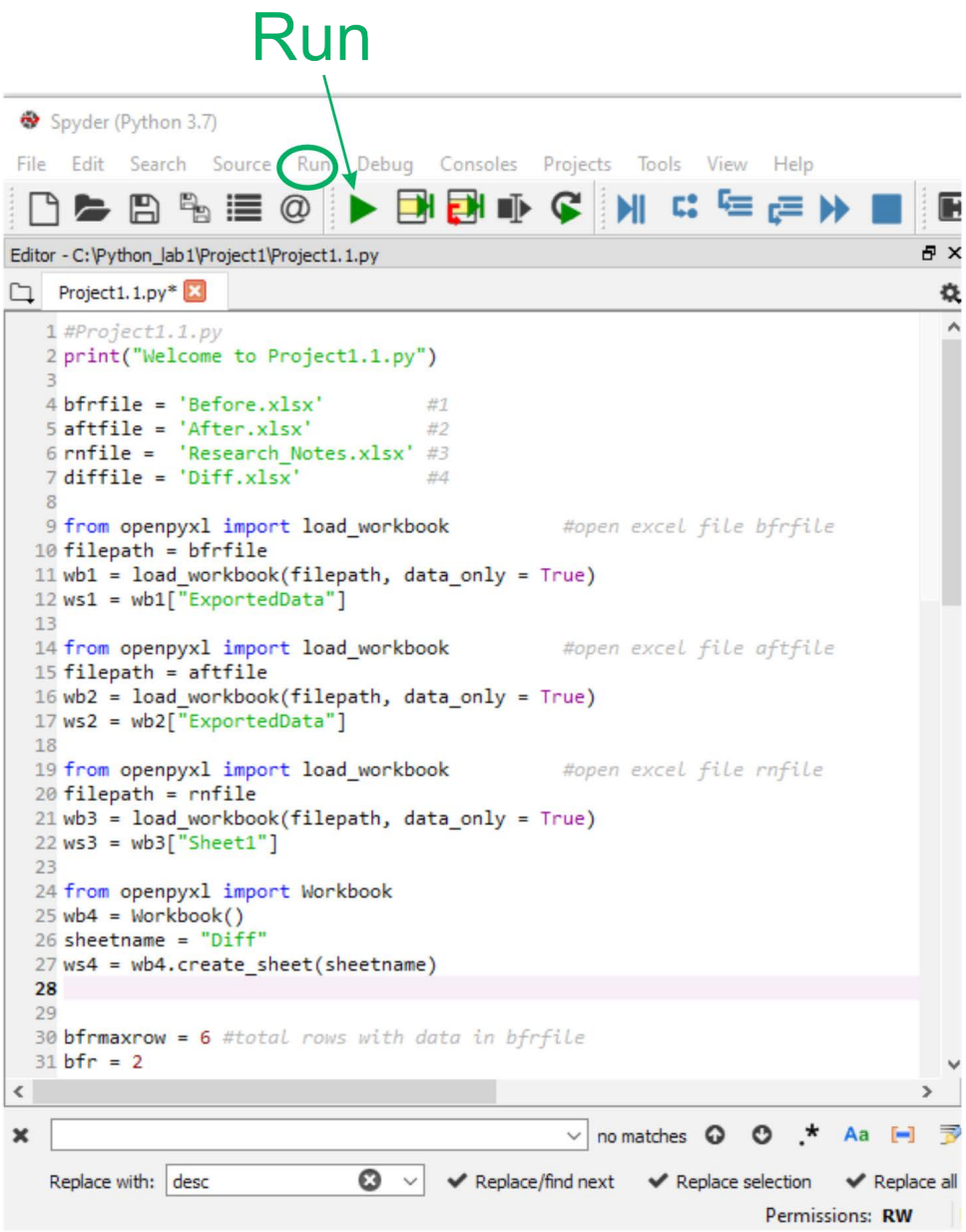


Figure 2.5 Run the Program

In the next figure, I have two panes open. The Editor is on the left, and the Output window is on the right. Initially, the Console window displays **In [1]:**. After I click "**Run**," the Console window changes, as shown below. The first output line displays the name of the program file and the working directory.

```
In [1]:  runfile('C:/Python_lab1/Project1/Project1.1.py', wdir='C:/Python_
         lab1/Project1')

Welcome to Project1.1.py

In [2]:
```

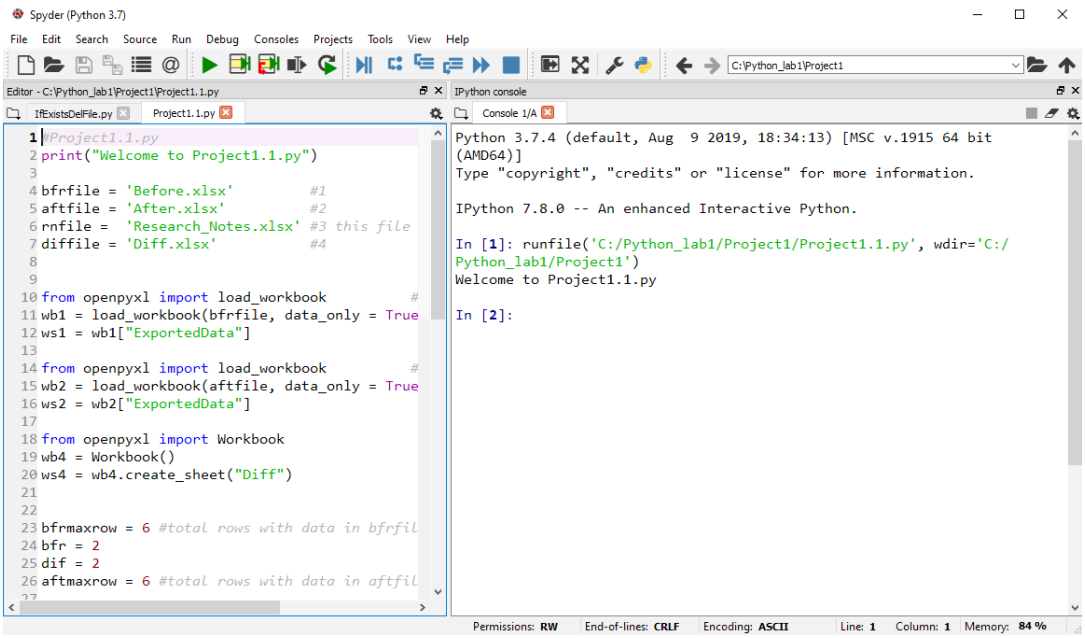


Figure 2.6 The Console

After running the program, there is a new file "*Diff.xlsx*" in the working directory, **wdir=**'C:/Python_lab1/Project1'.

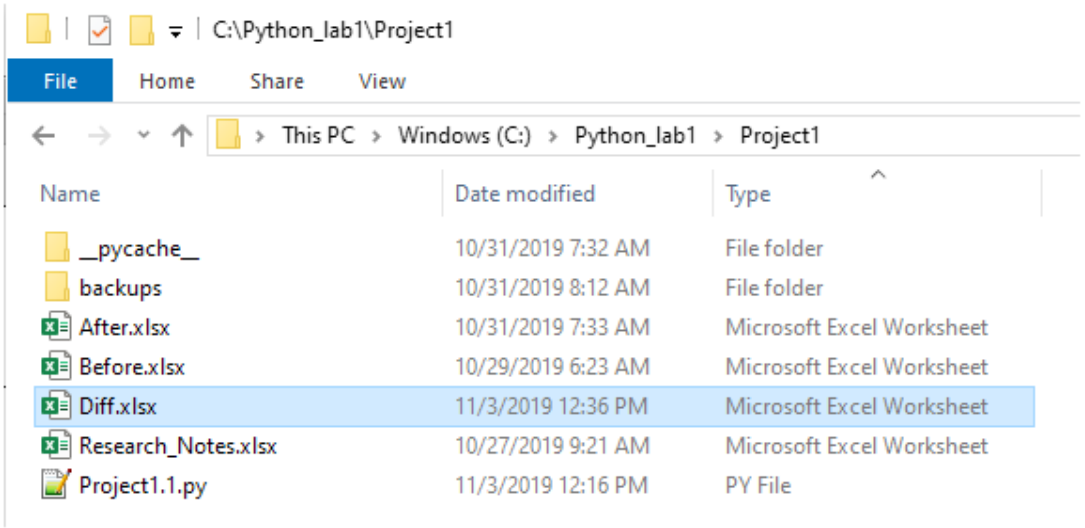


Figure 2.7 The New Diff.xlsx File

2.5 What's Next?

Your Lab environment is now setup. Let's move on to Chapter 3, where we look at the code line-by-line.

3. Project 1

In this chapter we discuss

Now that your environment is set up, we'll look at Project 1. While walking through the code, I'll introduce the following basic concepts, and explain each as it relates to the sample code. We'll look at chunks of code and focus on one particular concept at a time, so you can quickly skip over familiar concepts.

- Syntax
- Structure of Excel Files
- Variables and Data Types
- The openpyxl Library, Classes, & Objects
- Loop (Bfr counter)
- Read data from a Cell
- Loop (Aft counter)
- Search for a Match
- If ... Else Statements
- Comparison Operators

With a few minor adjustments, you can customize this code to work for your files. I'll show you where to add your Excel filenames and update column or row numbers, in the next Chapter.

3.1 Overview

In Lab 1 - Part 1, I compare data between two Excel files. This Lab is similar to a "vlookup" function. These are the basic steps, and I'll refer back to these steps throughout this Chapter.

Step 1. To begin, in the "*Before.xlsx*" file, compare the "**Item**" name in Column B for each row in the "*Before.xlsx*" Excel worksheet.

Step 2. Next, in Column B, find the row in the "*After.xlsx*" file with the same "**Item**" name from Step 1.

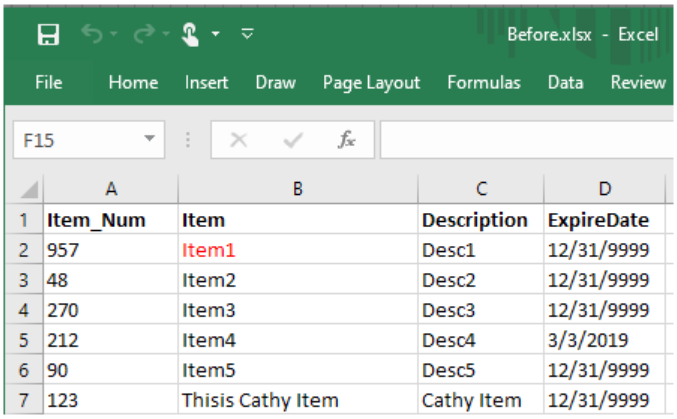
Step 3. Compare Column C "**Description**" in the "*Before.xlsx*" file with Column C "**Description**" in the "*After.xlsx*" file.

Step 4. If the data in Step 3 doesn't match, write the information to a new Excel file "*Diff.xlsx.*"

3.2 Structure of the Excel Files

Step 1. In the "*Before.xlsx*" file, I am using the "**ExportedData**" worksheet.

- In the code, Column B, "**Item**" is Column "2."
- In the code, Column C, "**Description**" is Column "3."



	A	B	C	D
1	Item_Num	Item	Description	ExpireDate
2	957	Item1	Desc1	12/31/9999
3	48	Item2	Desc2	12/31/9999
4	270	Item3	Desc3	12/31/9999
5	212	Item4	Desc4	3/3/2019
6	90	Item5	Desc5	12/31/9999
7	123	Thisis Cathy Item	Cathy Item	12/31/9999

Figure 3.1 Before.xlsx File

Step 2. In the "*After.xlsx*" file, I am using the "**ExportedData**" worksheet.

- In the code, Column B, "**Item**" is Column "2."
- In the code, Column C, "**Description**" is Column "3."

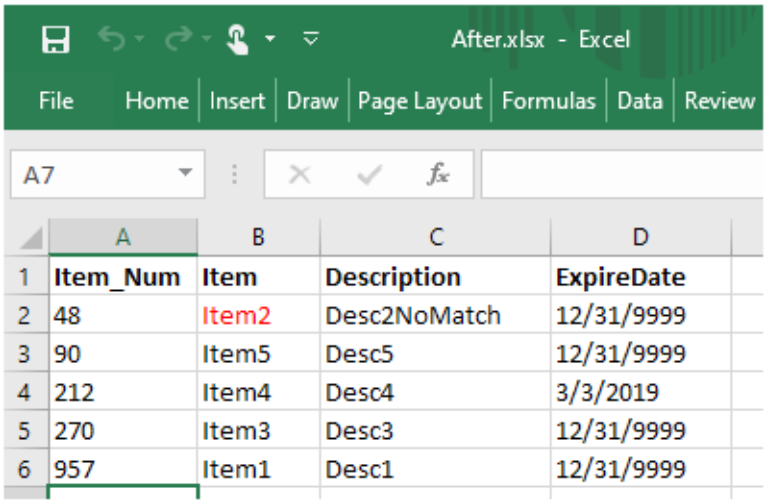


Figure 3.2 After.xlsx File

Step 4. In the new "*Diff.xlsx*" Excel workbook, I am writing to the "**Diff**" worksheet.

- In the new "*Diff.xlsx*" file, cell A2 is "**Item2**." In the code, this is "**Bfritem**" that represents Column "1" from "*Before.xlsx*."

- In the new "*Diff.xlsx*" file, cell B2 is "**Desc2**." In the code, this is "**Bfr Desc**" that represents Column "2" in "*Before.xlsx*."

- In the new "*Diff.xlsx*" file, C2 "**Aft Desc**" is from Column "C" from "*After.xlsx*."

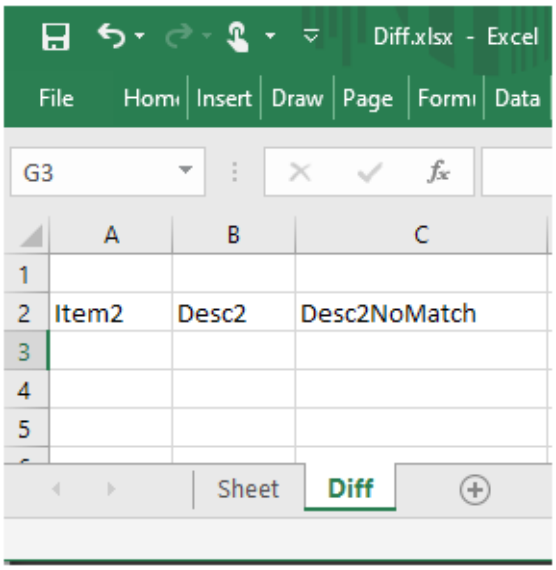


Figure 3.3 Diff.xlsx File

3.3 Syntax

A recommended best practice in Python is to use lowercase for variable and function names. Also, when creating variables or defining functions, do not use spaces. Instead, use underscores. Python has

reserved keywords like “global” or “try.” When you use a keyword as a variable name it causes a syntax error.

- Variable names begin with a letter.
- Use underscores instead of spaces.
- Use lowercase.
- Do not use keywords
- Numbers are allowed (except as the first character.)

Indentation in Python scripts defines a code block and must be consistent. You can see an example of the importance of indentation in the “Loop” topic.

3.4 Variables

Think of a variable as a container to store values. When a program runs, the value inside the variable may change. A letter or number is a value. An **assignment statement** creates a variable and assigns a value, as shown below. The left side of an assignment statement must be a variable.

```
>>>mynumber = 2000000
```

There are different types of values. In this lab we use these types:

- int
- string

Integers

When assigning integer values do not use commas. Python interprets 2,000,000 as three integers separated by commas. In the previous example, I assign 2000000 to the integer variable "**mynumber**."

Strings

To assign a value to a string variable, use single quotes, as shown below.

```
>>>myfilename = 'myExcelfile.xlsx'
```

A **string** is a sequence of characters. Python strings are immutable. You can not change an existing string, but you can create a new string with the modified data.

Note: A **list** is a collection of ordered values. A unique index number, or name, refers to each list item. The index is used when updating list items. Lists are discussed later in this Chapter.

Lab Variables

When translating the Lab steps into Python code, I've added variables to make it easier to work with the data.

Step 1. In the "*Before.xlsx*" file, compare the "**Item**" name from Column B for each row in that Excel file.

- The "**bfrfile**" variable represents the Excel filename. The type of this variable is a string.
- The "**bfr**" variable represents the current row number in the "*Before.xlsx*" Excel file. The type of this variable is an integer, and I use it to keep track of the row count.

- The "**bfrmaxrow**" variable represents the total number of rows in the "*Before.xlsx*" worksheet with data. The type of this variable is an integer.

The figure below assigns data to the **string** variables in lines 4 to 7. Notice the string data is enclosed in single quotes. The light grey text on the right side of some lines is a **comment**. Anything to the right of a hashtag # is a comment. Programmers may add comments to explain or clarify code.

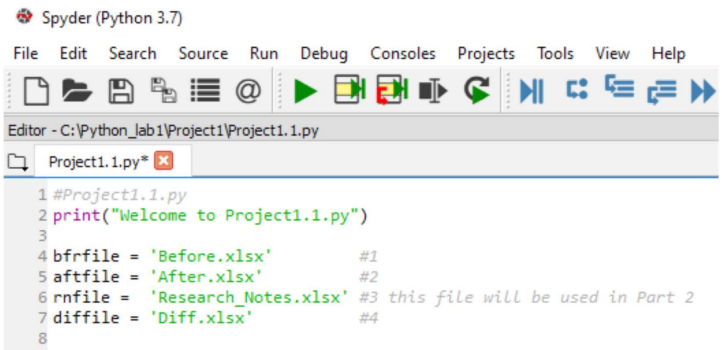


Figure 3.4 String Variables

Step 2. Find the row in the "*After.xlsx*" file in Column B, with the same "**Item**" name from Step 1. Loop through all rows in the "*After.xlsx*" file, checking for a match in Column B to Column B in "*Before.xlsx*."

- The "**aftfile**" variable represents the filename.
- The "**aft**" variable represents the current row number in "*After.xlsx*."
- The "**aftmaxrow**" variable represents the number of rows with data in the "*After.xlsx*" spreadsheet.

Step 3. After a match is found in Column B, compare Column C "**Description**" data in the "*Before.xlsx*" file with Column C "**Description**" data in the "*After.xlsx*" file.

Step 4. If the data in Step 3 doesn't match, write the information to a new Excel file "*Diff.xlsx.*"

- The "**diffile**" variable represents the "*Diff.xlsx*" filename.
- The "**dif**" variable represents the current row number in the "*Diff.xlsx*" file.

The figure below has additional variables on lines 23 to 26, and on line 30. These are all **integer** variables. When assigning integer values, simply type the number. Quotes are not needed.

```
22
23  bfrmaxrow = 6 #total rows with data in bfrfile
24  bfr = 2
25  dif = 2
26  aftmaxrow = 6 #total rows with data in aftfile
27
28  while bfr <= bfrmaxrow:
29      bfritem = wsl.cell(row = bfr, column = 2)
30      aft = 2
31      itemretired = 1
```

Figure 3.5 Integer Variables

3.5 Objects

An object is a collection of data (variables). With **openpyxl**, you assign **objects** to both the workbook and worksheet, and then you use those **objects** with **methods** to read or update values (the data).

```
from openpyxl import load_workbook
wb2 = load_workbook(aftfile, data_only = True)
ws2 = wb2["ExportedData"]
```

3.6 Lists

A **list** is a collection of ordered values. A unique index number, or name, refers to each list item. The index is used when updating list items.

```
ws1 = wb1["ExportedData"]
```

In this example, the workbook object "wb1" is a **list** item. The index "**ExportedData**" refers to one of the worksheets in the **list**.

3.7 The openpyxl Library

The **library** "openpyxl" is used to read and write Excel files. After a brief explanation of how to work with the library, we'll look at two of the openpyxl classes.

Step 1. Recall that for the "*Before.xlsx*" file, "**bfrfile**" represents that Excel file. The code below uses the "openpyxl" library to open the workbook for read access using the "load_workbook" method on line 11.

```
Line 10: from openpyxl import load_workbook
Line 11: wb1 = load_workbook(bfrfile, data_only = True)
Line 12: ws1 = wb1["ExportedData"]
```

In this example, the last line uses the workbook object "wb1" and references the **list** item "**ExportedData**". Recall that a **list** is a collection of ordered values. The index "**ExportedData**" refers to one of the worksheets in the **list**.

Description	Before.xlsx	After.xlsx	Diff.xlsx
Workbook	wb1 object	ws2 object	wb4 object
Worksheet	ws1 object "ExportedData"	ws2 object "ExportedData"	ws4 object "Diff"
Represents	"ExportedData"	"ExportedData"	"Diff"

Table 3.1 Excel Objects

To access data in an Excel workbook, first you assign **objects** to both the workbook and worksheet, and then you use those **objects** with **methods** to read or update values (the data). In the previous example, the first line of code opens the "openpyxl" library and imports the "**load_workbook**" **class**. On the second line, the "**load_workbook**" **method** creates a workbook **object** "**wb1**." The final step is to create a worksheet **object**, "**ws1**."

1. Import the "openpyxl" class "l**oad_workbook**."
2. On the line 11, the method "**load_workbook**" assigns the "**wb1**" object to the "**bfrfile**" file.
3. On the line 12, the object "**ws1**" is assigned to the sheet "**Exported Data**."

Note: Functions are a sequence of statements. A method is similar, except a **method** is used with objects.

Later in this Chapter, we'll use the worksheet object "**ws1**" with the "cell" **method** to read the cell data or "values."

Step 2. For the "*After.xlsx*" file, "**aftfile**" represents the file.

```
Line 14: from openpyxl import load_workbook
Line 15: wb2 = load_workbook(aftfile, data_only = True)
Line 16: ws2 = wb2["ExportedData"]
```

- In Step 2, the workbook **object** is "**wb2**."
- The worksheet **object** is "**ws2**."

Step 4. In Step 4, when the data from "*Before.xlsx*" column C does not match the data in "*After.xlsx*" column C, I write data into a new Excel file "*Diff.xlsx*." The code shown below creates new workbook and worksheet objects.

```
Line 18: from openpyxl import Workbook
Line 19: wb4 = Workbook()
Line 20: ws4 = wb4.create_sheet("Diff")
```

This code is slightly different than Step 2 and uses the "Workbook" **method** on line 19 to create the new "*Diff.xlsx*" workbook.

- In Step 4 the workbook **object** is "**wb4**."
- The worksheet **object** is "**ws4**."

At this point, we only need the workbook and worksheet **objects** (**wb4** and **ws4**.) Later in this example, we'll save this new workbook using the filename "**Diff.xlsx**." In case you're wondering what it looks like, this is the "save" **function** code.

```
wb4.save(diffile)
```

3.8 Loop (Bfr Counter)

This Project example uses two nested code "while statements." The actions that a program takes are called "statements." When the "while statement" on the first line is true, it executes a statement on the next line(s). In this example of a "while loop," I use two variables as counters for the loop.

- bfr
- bfrmaxrow

The first "while" statement on line 28 loops through the code while the statement on line 28 is true. Let's call this first "while loop" that begins on line 28 the "**bfr loop**."

```
28 while bfr <= bfrmaxrow:
```

On line 24 I set the variable **bfr** = **2**, and on line 23 we I **bfrmaxrow** = **6**. The statement above from line 28 evaluates to "**while 2 <= 6**."

Note: In the next Chapter, I'll show you a "**function**" to determine the maximum rows with data from your Excel file.

The "**bfr loop**" continues until the last line of code in the "**bfr loop**" on line 48. On line 48 I increment the bfr counter.

```
bfr = bfr + 1
```

The code loops from line 48 back to line 28, to begin the **second bfr loop**.

In the next figure, there is a red box around the "**bfr loop**" code from line 28 to 48. I added a red vertical dotted line to highlight where the code is indented.

Note the code on line 29, 30, 31, 32, and 48 is indented to the same vertical level, indicating it is all part of the same "**bfr loop**," and it runs as part of that loop. The shaded area is part of the second "while loop" (lines 33 to 46.)

Within the "**bfr loop,**" line 38 only runs when the **if statement** on line 37 is true. The "**bfr**" counter on line 48 is the last line in this "**bfr loop,**" and moves forward in the loop to the next row in the "*Before.xlsx*" spreadsheet.

Later we'll look at the nested "if statements" in lines 35 to 46. A nested if statement means there is a second "if statement" within the first.

Project1.1.py*

```
22
23 bfrmaxrow = 6 #total rows with data in bfrfile
24 bfr = 2
25 dif = 2
26 aftmaxrow = 6 #total rows with data in aftfile
27
28 while bfr <= bfrmaxrow:
29     bfritem = ws1.cell(row = bfr, column = 2)
30     aft = 2
31     itemretired = 1
32     while aft <= aftmaxrow:
33         #%%
34         aftitem = ws2.cell(row = aft, column = 2)
35         if bfritem.value == aftitem.value:
36             itemretired = 0
37             if ws1.cell(bfr, 3).value == ws2.cell(aft,  3 ).value: #cells match
38                 aft = aftmaxrow + 1
39             else:
40                 ws4.cell(row = dif, column = 1).value = bfritem.value
41                 ws4.cell(row = dif, column = 2).value = ws1.cell(row = bfr, column = 3).value
42                 ws4.cell(row = dif, column = 3).value = ws2.cell(row = aft, column = 3).value
43                 dif = dif + 1
44                 aft = aftmaxrow + 1
45         else:
46             aft = aft + 1 #move to next row in aftfile
47             #%%
48     bfr = bfr + 1    #move to next row in bfrfile
49
```

Figure 3.6 Bfr Loop

On line 24 I set the "**bfr**" counter = **2**. **Bfr** represents the row in the Excel "*Before.xlsx*" file. The last line of the "**bfr loop**" line 48 increments the "**bfr**" counter, in effect moving to the next row in the "*Before.xlsx*" workbook.

```
24 bfr = bfr + 1
```

In the next table, the ".value" column reflects the data in the Excel file.

Bfr Loop				
Bfr Loop	**Row Variable**	**Evaluates to:**	**Row bfr + Col B Represents**	**.Value**
1st time through	**bfr**	2	Before.xlsx B2	Item1
2nd time through	**bfr**	3	Before.xlsx B3	Item2

Table 3.2 Bfr Loop with Before.xlsx

Step 1 involves searching for the first item in **cell B2** in the "*Before.xls*" file. The first time the code runs through the "**bfr loop**" and "**bfr**" = 2, cell B2 is "Item1."

Before.xlsx		
	Excel	**in Code**
Workbook	Before.xlsx	wb1 object
Worksheet	ExportedData	ws1 object
Cell	B2	bfritem
Row	2	bfr

Table 3.3 Before.xlsx Excel Code

The "**aft loop**" continues looping and moves through all the rows in the "*After.xlsx*" file.

After line 48 runs **bfr** = 3 and the "**bfr loop**" continues and moves back to line 28. On line 29, the object bfritem.value represents cell B3 or "Item2" in the "*Before.xlsx*" file. In effect, the code moves to the next row of data in the "*Before.xls*" Excel file. The program now runs through the second loop of the "**bfr loop**."

3.9 Read Data from a Cell

Within the first **While Loop** that we're calling the "**bfr loop**," let's look at line 29. Earlier, I pointed the object "**wb1**" to the workbook "*Before.xlsx*," and set the "**ws1**" object to the "**Exported Data**" sheet in that workbook "*Before.xlsx*."

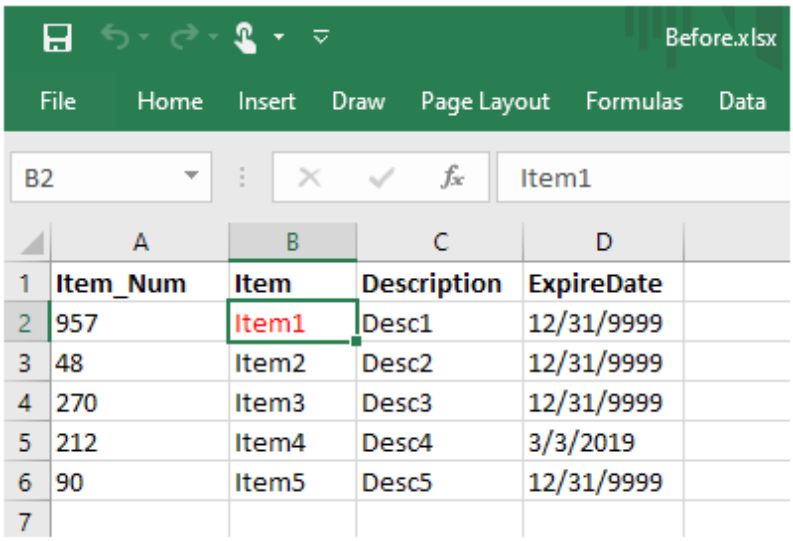
The first time the code runs through the loop **bfr** = 2. The code on line 29, shown below, assigns the "**bfritem**" object to cell B2 in the "*Before.xls*" Excel file.

```
29 bfritem = ws1.cell(row = bfr, column = 2)
```

Bfr Loop			
Bfr Loop	**Object**	**Row bfr + Col B Represents**	**bfritem.Value**
1st time through	bfritem	Before.xlsx B2	Item1
2nd time through	bfritem	Before.xlsx B3	Item2

Table 3.4 Bfr Loop with bfritem

Line 35 uses the "**.value**" method with the "bfritem" object to compare the the value to the aftitem.value. Line 37 demonstrates another way to use the "**.cell**" method with the "ws1" worksheet object for a similar comparison. Later, in line 40, I use the .cell method to assign a new value to a cell in the new worksheet.



	A	B	C	D
1	Item_Num	Item	Description	ExpireDate
2	957	Item1	Desc1	12/31/9999
3	48	Item2	Desc2	12/31/9999
4	270	Item3	Desc3	12/31/9999
5	212	Item4	Desc4	3/3/2019
6	90	Item5	Desc5	12/31/9999
7				

Figure 3.7 Bfr cell B2

The second time the code loops through the "**bfr loop**" code, **bfr** = 3, and the code on line 29 evaluates to "Item2."

3.10 Loop (Aft Counter)

Earlier, I mentioned there are nested loops and if statements in this code. In the earlier figure, the shaded area represents a second nested while loop. Let's call this second loop that begins on line 33 the "**aft loop**." The "**aft loop**" begins on line 33, shown below, and goes through line 46.

```
33 while aft <= aftmaxrow:
```

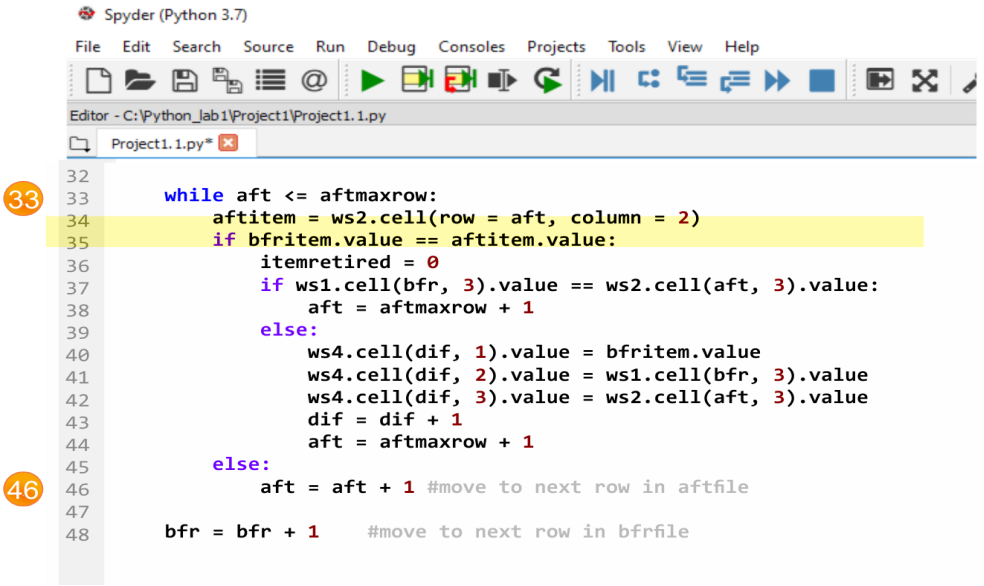


Figure 3.8 The "Aft Loop" from line 33 to 46

Aft Loop				
Bfr Loop	**Row Variable**	**aft counter**	**Row aft + Col B Represents**	**Aftitem.Value**
1st time through	aft	2	After.xlsx B2	Item2
2nd time through	aft	3	After.xlsx B3	Item5

Table 3.5 Aft Loop with After.xlsx

In the same way we set a counter **bfr** = **2** on line 24, I am setting the counter **aft** = **2** on line 30, as shown below. Since this is a nested loop, it's important to set this counter **aft** = **2** **within** this second "**bfr**

loop." Because line 30 is indented, it's "within" the while "**bfr loop**" I started on line 28.

```
aft = 2
```

```
27
28 while bfr <= bfrmaxrow:
29     bfritem = ws1.cell(row = bfr, column = 2)
30     aft = 2
31     itemretired = 1
32
```

Figure 3.9 Aft Counter Line 30

The first loop runs line 29 and in effect points the "bfritem" to cell B2 in the "*Before.xlsx*" file.

3.11 Loop Until Column 2 Match

Within the "**aft loop**" that starts on **line** 33, we compare Column B values on Line 35. Step 2 involves searching for the "**bfritem**" in the "*After.xls*" file.

```
if bfritem.value == aftitem.value:
```

Line 34, shown below, sets the "**aftitem**" object to cell B2 in the "*After.xls*" file when the code loops through the first time.

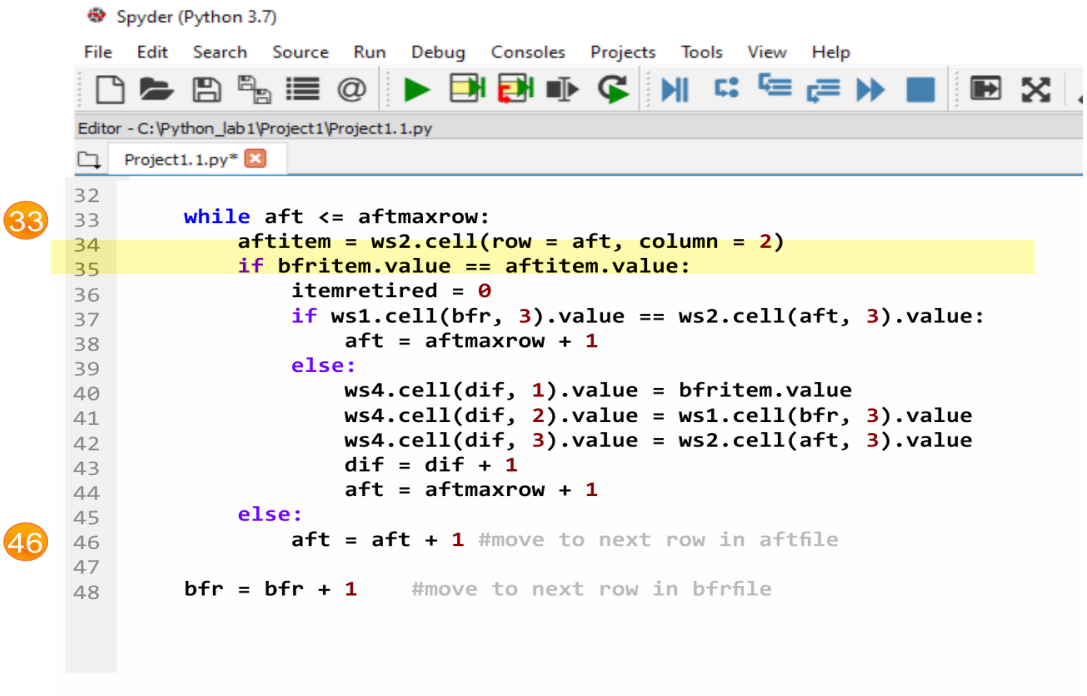


Figure 3.10 Lines 33 to 46

As shown in the next table, the first time Python loops through the code, the "**bfritem**" has a value of "**Item1**" from cell B2 of the "*Before.xlsx*" file. Then, line 34 sets the **aftitem** to ws2.cell(aft, 2).

```
aftitem = ws2.cell(row = aft, column = 2)
```

When the code loops through the first time, in line 35 the **bfritem** value of "**Item1**" does not equal the "**aftitem**" value "**Item2**."

```
35 if bfritem.value == aftitem.value:
```

The Aft Loop						
Aft Loop	**Before.xlsx cell**	**After.xlsx cell**	**bfr counter**	**aft counter**	**bfritem. value**	**aftitem. value**
1st loop	B2	B2	2	2	**Item1**	Item2
2nd loop	B2	B3	2	3	Item1	Item5
3rd loop	B2	B4	2	4	Item1	Item4
4th loop	B2	B5	2	5	Item1	Item3
5th loop	B2	B6	2	6	Item1	Item1

Table 3.6 Inside the Aft Loop

Because the two values in **Column B** do not match the first time through the loop, the "**aft loop**" increments the aft counter in line 46 of the code, as shown below. In effect, this code moves the aft counter to the next row of data in the "*After.xlsx*" file.

```
46 aft = aft + 1
```

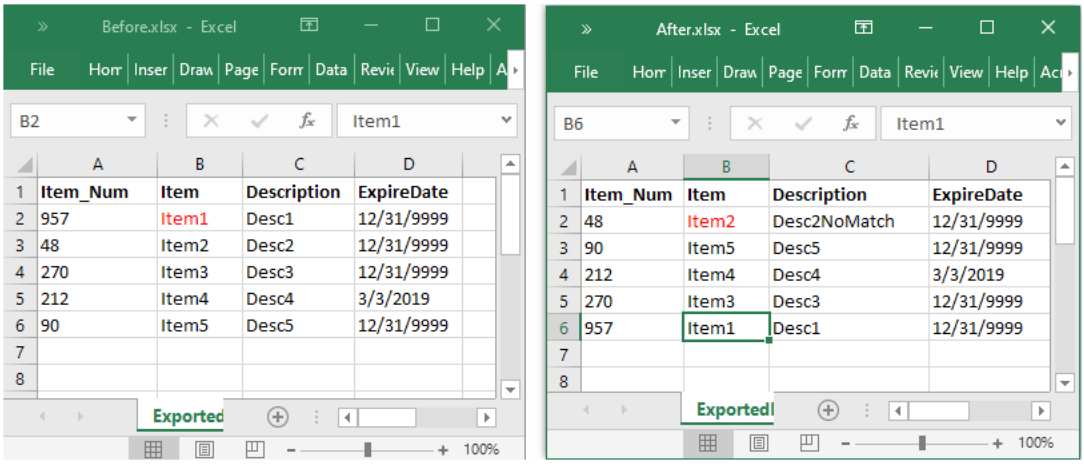


Figure 3.11 Compare B2 Cells

The code continues to loop and runs to line 46 and increments the "**aft**" counter to 3, then loops back to line 33. This variable represents row 3 in the "*After.xlsx*" file. The code on line 33 evaluates to 3 <= 6, and the loop continues.

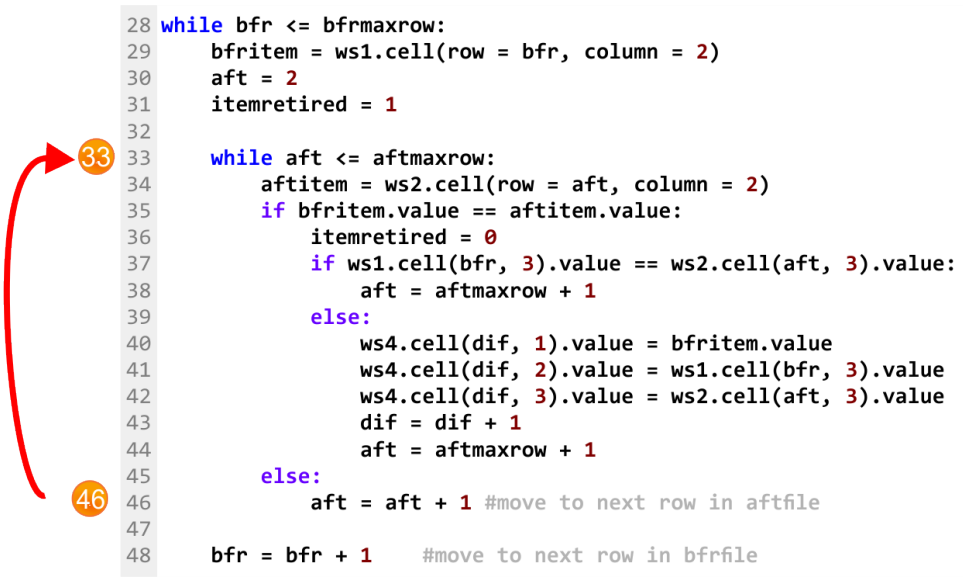
```
33 while aft <= aftmaxrow:
```

The code continues to loop from line 46 back up to line 33, incriminating the aft counter.

The Code Runs and Loops Through the Lines

The following chart is a visual representation of how the code loops back and forth as it runs, branching to different lines of code. Follow along as the code steps into while loops and if statements, as shown in the second column, "Line." The first column represents the **Loop** or **if statement**, and you can see the code moves back and forth.

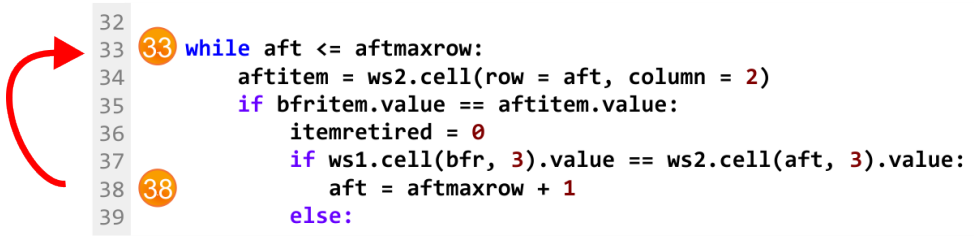
bfrmaxrow =6, aftmaxrow = 6			
	Line	**evaluates to**	**Notes**
1st bfr loop, 1st aft loop	28	2<=6	bfr=2, aft =2, true go to line 29
1st bfr loop, 1st aft loop	33	2 <= 6	bfr=2, aft =2, true go to line 34
1st bfr loop, 1st aft loop	35	item1==item2	Not true, go to line 45
1st bfr loop, 1st aft loop	45	else	
1st bfr loop, 1st aft loop	46	aft = aft +1	bfr = 2, aft is now 3, **go back to line 33**




```
28 while bfr <= bfrmaxrow:
29     bfritem = ws1.cell(row = bfr, column = 2)
30     aft = 2
31     itemretired = 1
32
33     while aft <= aftmaxrow:
34         aftitem = ws2.cell(row = aft, column = 2)
35         if bfritem.value == aftitem.value:
36             itemretired = 0
37             if ws1.cell(bfr, 3).value == ws2.cell(aft, 3).value:
38                 aft = aftmaxrow + 1
39             else:
40                 ws4.cell(dif, 1).value = bfritem.value
41                 ws4.cell(dif, 2).value = ws1.cell(bfr, 3).value
42                 ws4.cell(dif, 3).value = ws2.cell(aft, 3).value
43                 dif = dif + 1
44                 aft = aftmaxrow + 1
45         else:
46             aft = aft + 1 #move to next row in aftfile
47
48     bfr = bfr + 1    #move to next row in bfrfile
```

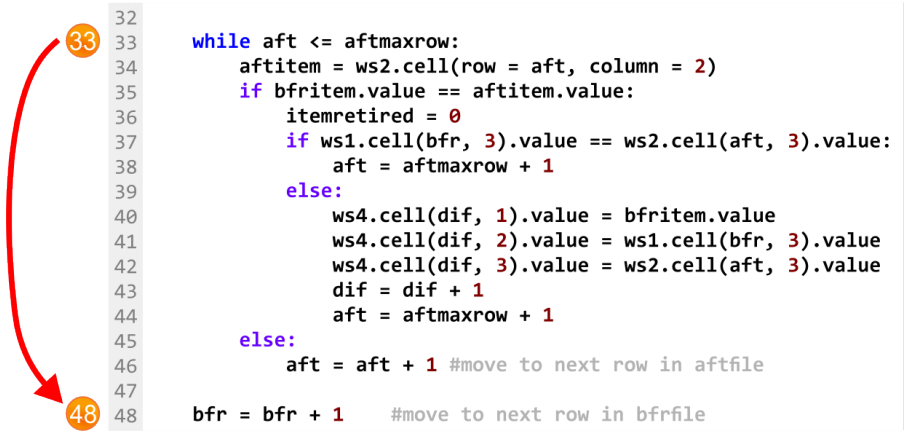
bfrmaxrow =6, aftmaxrow = 6			
	Line	**evaluates to**	**Notes**
1st bfr loop, 2nd aft loop	33	3 <=6	bfr=2, aft =3, true go to line 34
1st bfr loop, 2nd aft loop	35	item1==item5	Not true, go to line 45
1st bfr loop, 2nd aft loop	45	else	…
1st bfr loop, 2nd aft loop	46	aft = aft +1	bfr = 2, aft is now 4, go back to line 33
1st bfr loop, 3rd aft loop	33	4 <=6	bfr=2, aft =4, true go to line 34
1st bfr loop, 3rd aft loop	35	item1==item4	Not true, go to line 45
1st bfr loop, 3rd aft loop	45	else	…
1st bfr loop, 3rd aft loop	46	aft = aft +1	bfr = 2, aft is now 5, go back to line 33
1st bfr loop, 4th aft loop	33	5 <=6	bfr = 2, aft =5, true go to line 34
1st bfr loop, 4th aft loop	35	item1==item3	Not true, go to line 45
1st bfr loop, 4th aft loop	45	else	

bfrmaxrow =6, aftmaxrow = 6			
	Line	**evaluates to**	**Notes**
1st bfr loop, 4th aft loop	46	aft = aft +1	bfr = 2, aft is now 6, go back to line 33
1st bfr loop, 5th aft loop	33	5 <=6	bfr=2, aft =6
1st bfr loop, 5th aft loop	35	item1 ==item1	true, go to line 38
1st bfr loop, 5th aft loop	38	6 = 6 +1	bfr =2, aftmaxrow = 7, go back to line 33

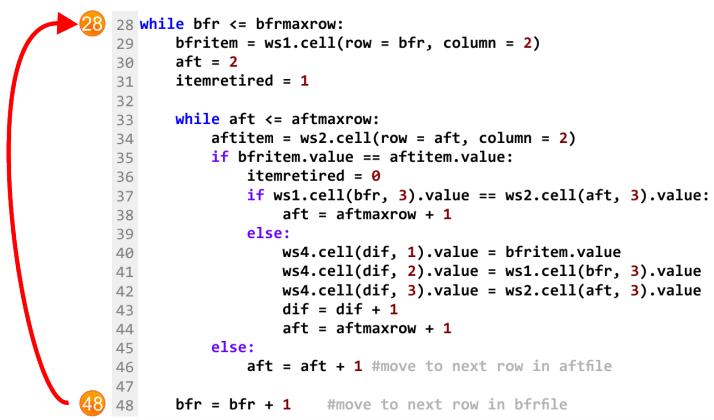


```
32
33  while aft <= aftmaxrow:
34      aftitem = ws2.cell(row = aft, column = 2)
35      if bfritem.value == aftitem.value:
36          itemretired = 0
37          if ws1.cell(bfr, 3).value == ws2.cell(aft, 3).value:
38              aft = aftmaxrow + 1
39          else:
```

1st bfr loop, 5th aft loop	33	7<=6	Not true, go down to line 48



```
32
33  while aft <= aftmaxrow:
34      aftitem = ws2.cell(row = aft, column = 2)
35      if bfritem.value == aftitem.value:
36          itemretired = 0
37          if ws1.cell(bfr, 3).value == ws2.cell(aft, 3).value:
38              aft = aftmaxrow + 1
39          else:
40              ws4.cell(dif, 1).value = bfritem.value
41              ws4.cell(dif, 2).value = ws1.cell(bfr, 3).value
42              ws4.cell(dif, 3).value = ws2.cell(aft, 3).value
43              dif = dif + 1
44              aft = aftmaxrow + 1
45      else:
46          aft = aft + 1 #move to next row in aftfile
47
48  bfr = bfr + 1    #move to next row in bfrfile
```

bfrmaxrow =6, aftmaxrow = 6			
	Line	**evaluates to**	**Notes**
1st bfr loop, 5th aft loop	48	2 = 2 +1	bfr =3, go back to line 28
			
2nd bfr loop	28	3 <= 6	bfr = 3, aft = 6, true go to line 29
2nd bfr loop	30	aft = 2	reset aft counter to begin a new "aft loop"
2nd bfr loop, 1st aft loop	33	2 <= 6	bfr = 3, aft = 2, true go to line 34

```
28 while bfr <= bfrmaxrow:
29     bfritem = ws1.cell(row = bfr, column = 2)
30     aft = 2
31     itemretired = 1
32
33     while aft <= aftmaxrow:
34         aftitem = ws2.cell(row = aft, column = 2)
35         if bfritem.value == aftitem.value:
36             itemretired = 0
37             if ws1.cell(bfr, 3).value == ws2.cell(aft, 3).value:
38                 aft = aftmaxrow + 1
39             else:
40                 ws4.cell(dif, 1).value = bfritem.value
41                 ws4.cell(dif, 2).value = ws1.cell(bfr, 3).value
42                 ws4.cell(dif, 3).value = ws2.cell(aft, 3).value
43                 dif = dif + 1
44                 aft = aftmaxrow + 1
45         else:
46             aft = aft + 1 #move to next row in aftfile
47
48     bfr = bfr + 1    #move to next row in bfrfile
```

Table 3.7 Looping Through Code

A Column B Match is Found

The fifth time the loop runs, **aft = 6** and the "After.xlsx" B2 cell on row 6 is "Item 1." At this point, a match is found on line 35, which compares the two cells in Column B, as shown below.

```
35 if bfritem.value == aftitem.value
```

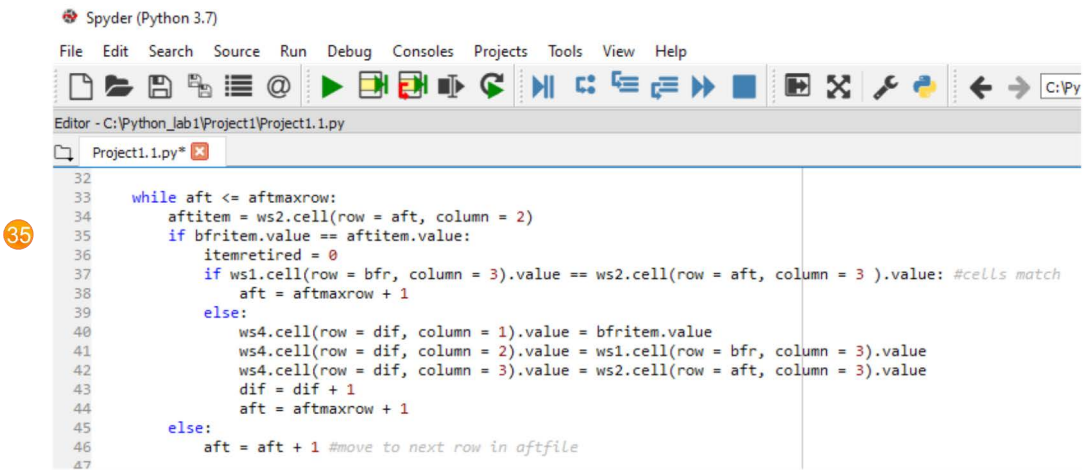


Figure 3.12 Compare on Line 35

Column C Data Does Not Match

Now, line 37 compares the values in **Column C**, as shown below. Notice I shortened the syntax from **cell(row=bfr, column =3)** to **cell(bfr, 3)**.

```
37 if ws1.cell(bfr, 3).value == ws2.cell(aft, 3 ).value:
```

Now, the code runs the second "**Bfr loop**." At this point, Bfr=3 and Aft=2. The code evaluates the "if statement" on line 37 and the Column C data does NOT match. Now the code jumps to the "else" statement on line 39 runs, which writes data to the output "*Diff.xlsx*" file.

Note: Another way to write this part of the code would be to use the comparison operator **not equal to !=**.

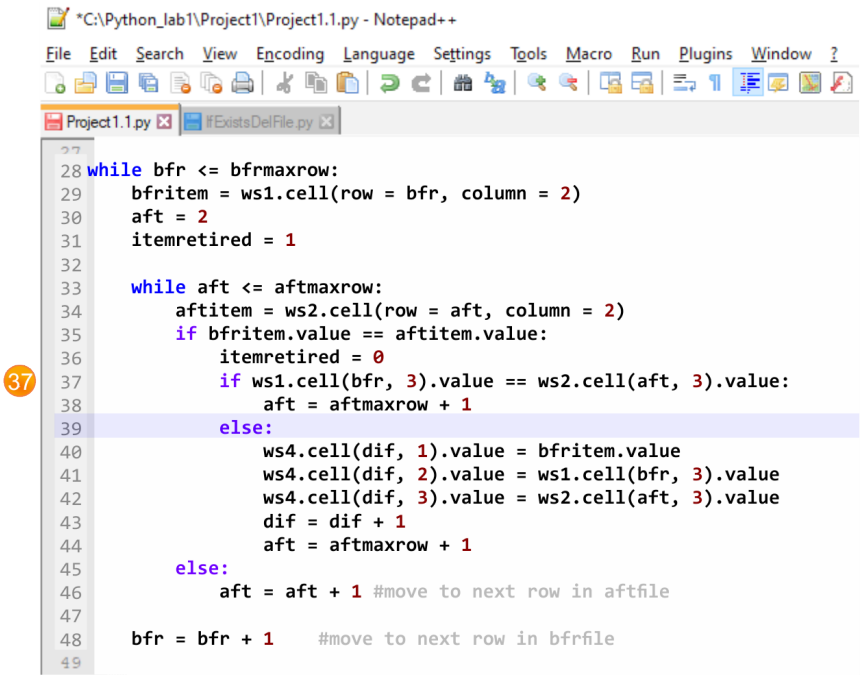


Figure 3.13 Column C Data does NOT Match

Let's take a moment to look at the code and output. The table below shows the actual code, as well as what it will evaluate to. When I say "evaluate to," this is what will be written into the "*Diff.xlsx*" file.

Code	Excel File	Excel Column	Column Counter	Row Counter	Value
ws4.cell(diff,1).value	Diff.xlsx	A2	1	2	Item2
bfritem.value	Before.xlsx	B3	2	3	Item2
ws4.cell(diff,2).value	Diff.xlsx	B2	2	2	Desc2
ws1.cell(bfr,3).value	Before.xlsx	C3	3	3	Desc2
ws4.cell(diff,3).value	Diff.xlsx	C2	3	2	Desc2NoMatch
ws2.cell(diff,3).value	After.xlsx	C2	3	2	Desc2NoMatch

Table 3.8 Data for the Diff.xlsx Output File

A quick look at the Excel files shows how the data flows from the Excel files into the new "*Diff.xlsx*" file.

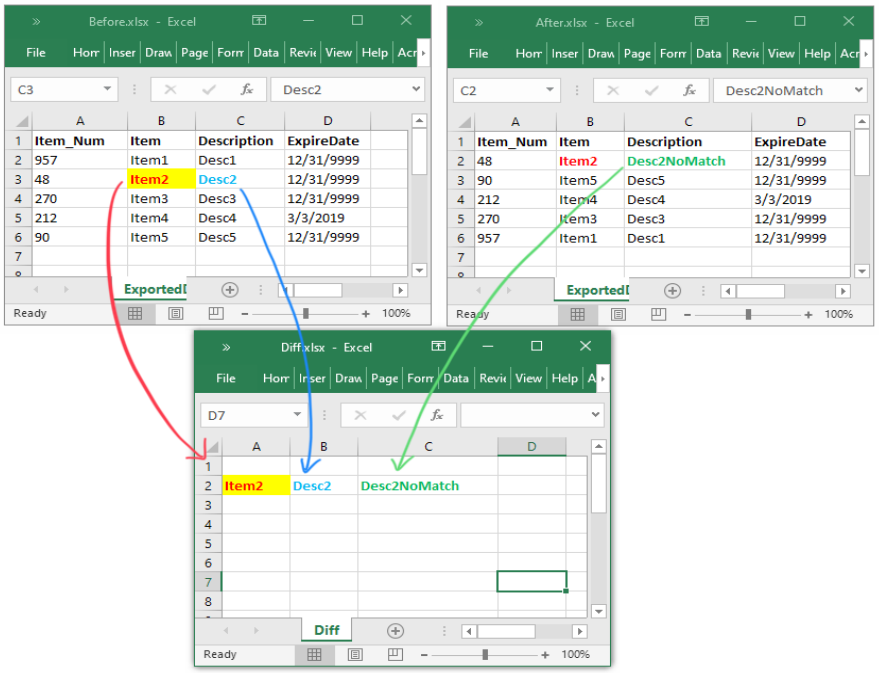


Figure 3.14 Data Flows to Diff.xlsx Output File

The "else:" code on line 39 creates the output data in lines 40 to 42 for the "*Diff.xlsx*" file, using the ws4.cell object reference.

```
37 if ws1.cell(bfr, 3).value == ws2.cell(aft,  3).value:
       aft = aftmaxrow + 1
39 else:
40     ws4.cell(dif, 1).value = bfritem.value
       ws4.cell(dif, 2).value = ws1.cell(bfr, 3).value
       ws4.cell(dif, 3).value = ws2.cell(aft, 3).value
```

Next, line 46 increments the **dif** counter to move to the next row in the "*Diff.xlsx*" output file, as shown below. The next "difference" we find would write to this row.

```
46     dif = dif + 1
```

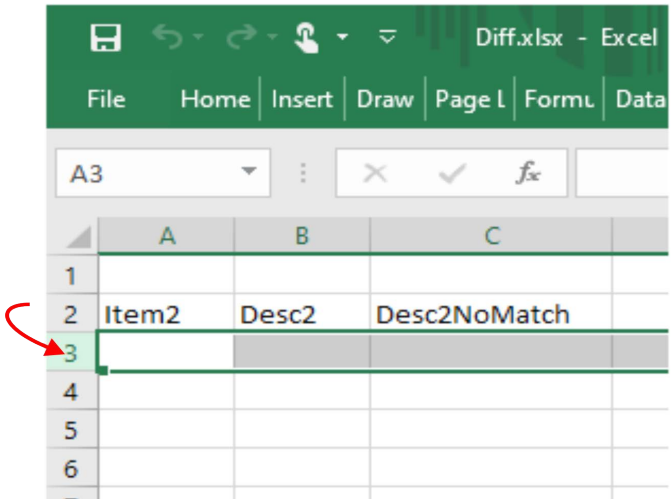


Figure 3.15 Next Row in Diff.xlsx Spreadsheet

The "**aft loop**" found a match, and I want to exit the "**aft loop**" and move to the next item in the "**bfr loop**." Line 44 sets the **aft** counter to a value greater than **aftmaxrow**, as shown below.

```
44    aft = aftmaxrow + 1
```

The code exits the "**aft loop**"on line 44 and moves back up to line 33. When the code evaluates **aft** on line 33, **aft** is **>** **aftmaxrow**. The code now skips from line 33 back down to line 48.

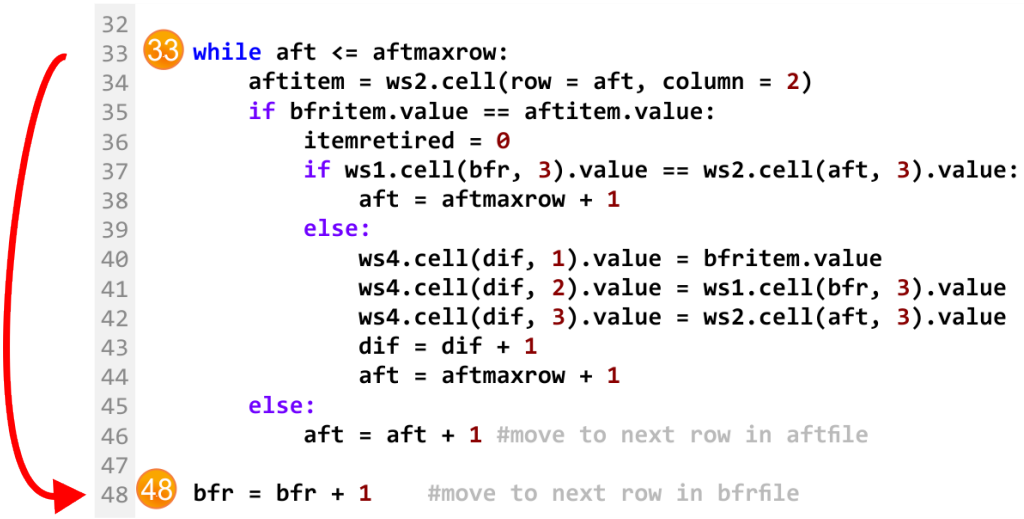
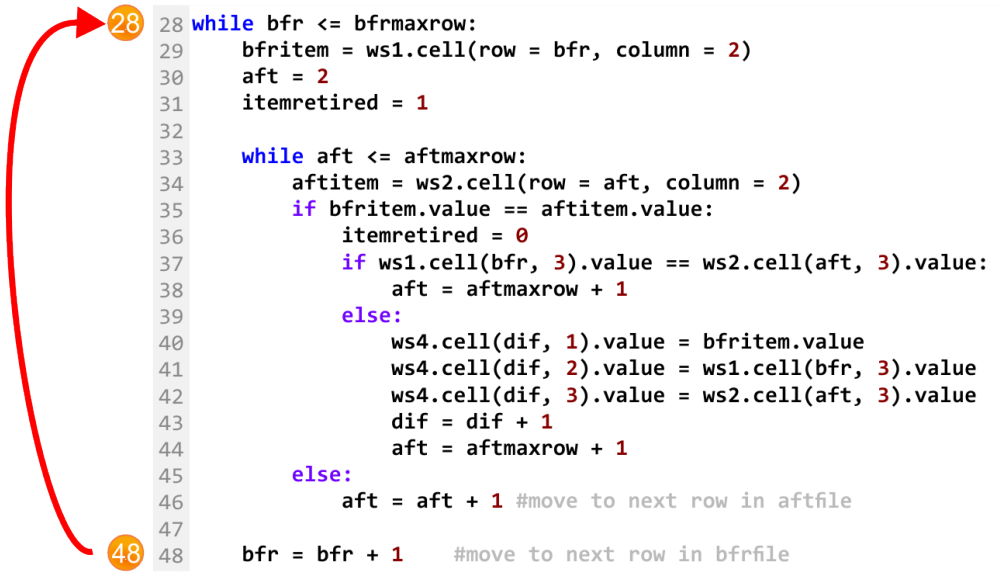


Figure 3.16 Line 33 moves to Line 48

On line 48 the **bfr** counter is incriminated as shown below.

```
48  bfr = bfr + 1
```

Next, the code loops from line 48 back to line 28, effectively moving to the next row in the "*Before.xlsx*" spreadsheet.



```
28 while bfr <= bfrmaxrow:
29     bfritem = ws1.cell(row = bfr, column = 2)
30     aft = 2
31     itemretired = 1
32
33     while aft <= aftmaxrow:
34         aftitem = ws2.cell(row = aft, column = 2)
35         if bfritem.value == aftitem.value:
36             itemretired = 0
37             if ws1.cell(bfr, 3).value == ws2.cell(aft, 3).value:
38                 aft = aftmaxrow + 1
39             else:
40                 ws4.cell(dif, 1).value = bfritem.value
41                 ws4.cell(dif, 2).value = ws1.cell(bfr, 3).value
42                 ws4.cell(dif, 3).value = ws2.cell(aft, 3).value
43                 dif = dif + 1
44                 aft = aftmaxrow + 1
45         else:
46             aft = aft + 1 #move to next row in aftfile
47
48     bfr = bfr + 1    #move to next row in bfrfile
```

Figure 3.17 Line 48 moves to Line 28

The code continues to loop until there is a match for Item 2. At that pointt, he code compares the "Description" column 3 values. In line 40, I use the **.cell** method to assign a new value to a cell in the new worksheet.

Diff.xlsx			
	ws4.cell(diff,1) or Column 1	**ws4.cell(diff,2) or Column 2**	**ws4.cell(diff,3) or Column 3**
Code	bftitem.value	ws1.cell(Bfr, 3).value	ws2.cell(aft, 3).value
Evaluates to:	Item2	Desc2	Desc2NoMatch

Table 3.9 The Diff.xlsx File

3.12 Save the New Excel File

Eventually, the program ends. The last line uses the .save method to save the new Excel file.

```
wb4.save(diffile)
```

Customize Project 1

In this chapter we discuss

Delete Worksheets

Count Rows With Data

Search for Strings & Substrings

Delete Rows

Headings and Formatting

Delete Existing File

Adding Your Filename, Worksheet, & Column

Item Retired

New Items

Compare Dates

Now that the basic Project 1 program is working, I want to revisit the script to simplify and improve the code. Did you notice the output

Excel file is a little drab? It would definitely benefit from headings and formating.

So far the Excel sample data files were relatively simple without a lot of extraneous data. Unfortunately, this rarely seems to be the case with a new project. The files always need some type of data scrubbing. Part of this lab will search for data to delete rows I don't want in my data set. We'll also look at adding methods to automate basic tasks.

- Delete worksheets
- Count rows with data in the Excel file
- Search strings
- Delete rows

Finally, I'll show you how to customize the code to make it your own. This involves using your Excel filenames, and the particular columns you want to compare.

4.1 Delete Worksheets

When I created a new workbook, the worksheet "Sheet" was created by default. On line 23, I delete that worksheet using the function "**get_sheet_by_name**."

wb4.remove(wb4.get_sheet_by_name(name = 'Sheet'))

```
19
20 from openpyxl import Workbook
21 wb4 = Workbook()
22 ws4 = wb4.create_sheet("Diff")
23 wb4.remove(wb4.get_sheet_by_name(name = 'Sheet'))
24
```

Figure 4.1 Delete Worksheet

Note: Functions are a sequence of statements. A method is similar, except a **method** is used with objects.

4.2 Count Rows With Data

In the original version of the program, I provided a value for the total rows in the Excel spreadsheet. A better way to set the value is to use the function "**max_row**." Max_row returns the maximum row index containing data.

```
bfrmaxrow = ws1.max_row
```

```
24
25 bfrmaxrow = ws1.max_row
26 bfr = 2
27 dif = 2
28 aftmaxrow = 6 #total rows with data in aftfile
29
30 while bfr <= bfrmaxrow:
31     bfritem = ws1.cell(row = bfr, column = 2)
32     aft = 2
33     itemretired = 1
```

Figure 4.2 max_row Function

4.3 Search for Strings & Substrings

This section contains two methods for searching for a string or substring.

- .search
- .find

Search

In the script **Project1_Part2_3.py**, I search for the phrase "Discontinued" in column "C" of the Excel file "*Dates.xlsx*." When I create a workbook object for the Excel file "*Dates.xlsx*" on line 8, I do not use "data_only = **true**," because I want to change the file.

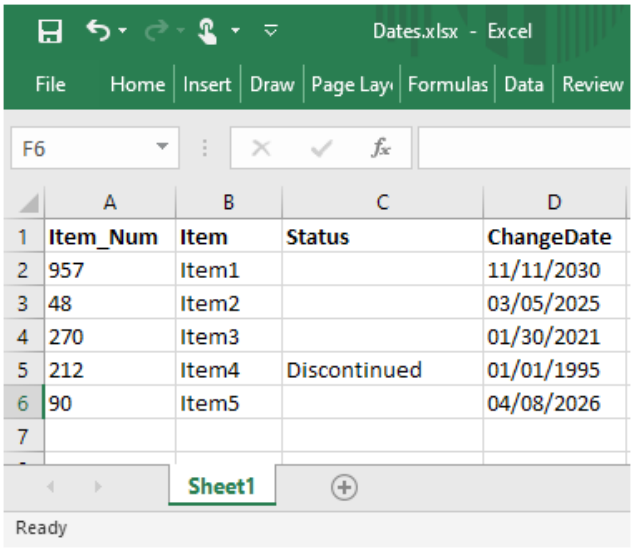


Figure 4.3 The Excel File "Dates.xlsx"

In the script **Project1_Part2_3.py**, line 5 imports the module "**re**." The "**re**.search" method looks for the string "Discontinued" in line 16.

In row 11, I set a counter "**r**" to 2, because I want to skip the headings in row 1. The "while" loop from lines 13 to 19 moves through each row in the "Dates.xlsx" file.

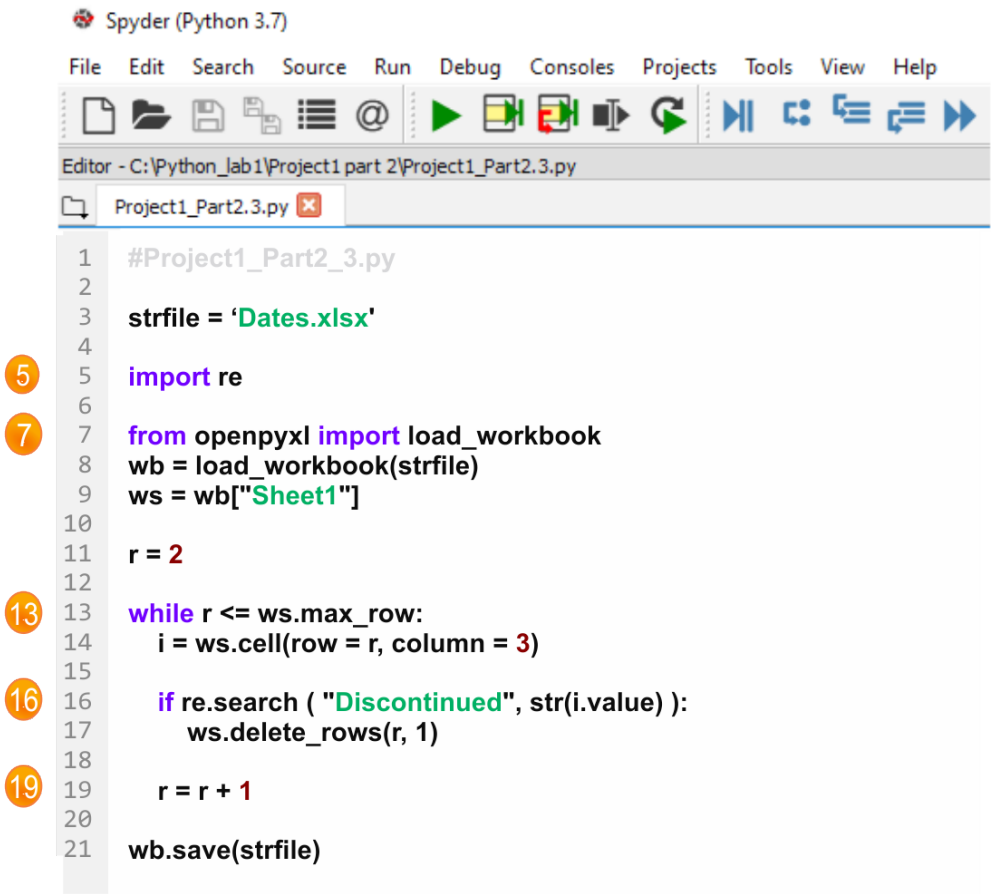


Figure 4.4 Search and Delete Row

Find

In the script **Project1_Part2_7.py**, I search for a substring within a string on line 19, using the "**find**" method. The "**find**" method returns **-1** if the substring is not found. If the substring is found, the

"**find**" method returns the location within the string. In line 19 in the next figure, the variable "**foundit**" is set to the location within the string.

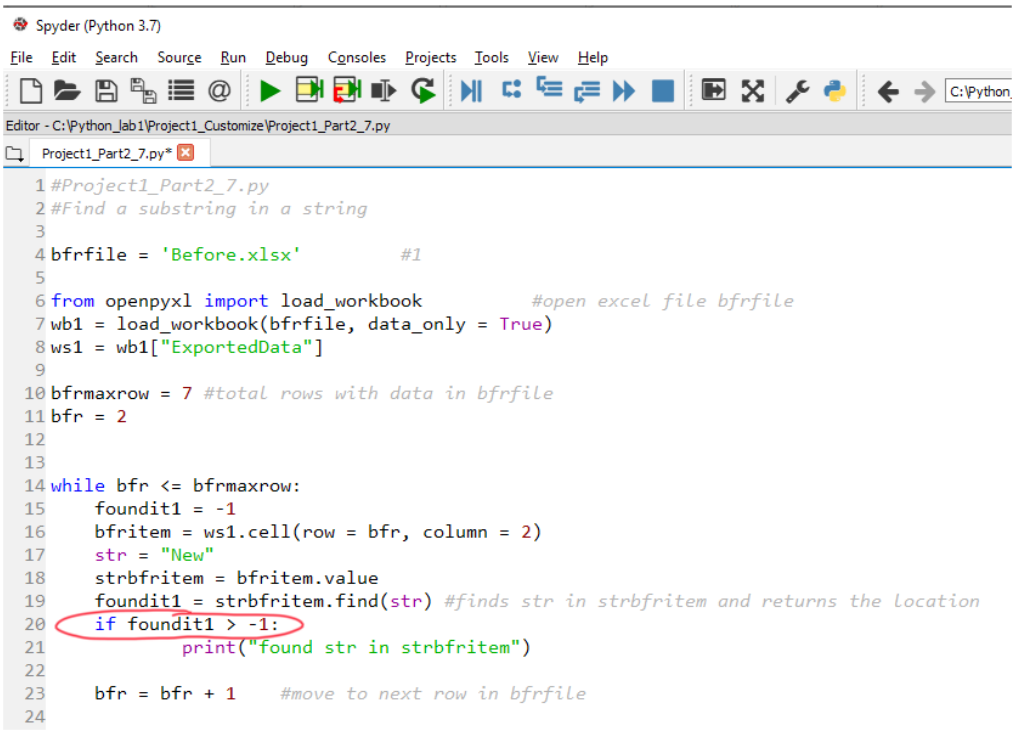


Figure 4.5 Search for Substring

In this example, I search for the substring "**New**" within Column 2 of the "*Before.xlsx*" file.

4.4 Delete Rows

As part of my data scrubbing, I delete rows that have the phrase "Discontinued" in column C. In the previous script **Project1_Part2_3.py**, the method "**delete_rows**" is available in the

latest version of **openpyxl**. When the "if" statement on line 16 is true, the code moves to line 17 and deletes the row.

4.5 Headings and Formatting

In the same way I used the "openpyxl" library, I imported my own module called "**FormatHeadings.py**." First, let's look at the code in "**FormatHeadings.py**." Line 1 defines the "**fh**" method. Notice line 1 begins with "**def**" and ends in a colon **:**. The method "**fh**" takes five arguments, as shown below.

```
def fh(ws4, str1, str2, str3, colwidth):
```

The next step would be to add the code below to the script file, for example **Project1.1.py**. This script imports my module, and then calls my method "**fh**." .

```
from FormatHeadings import fh

fh(ws4, "Item", "Description", "Comments", 20)
```

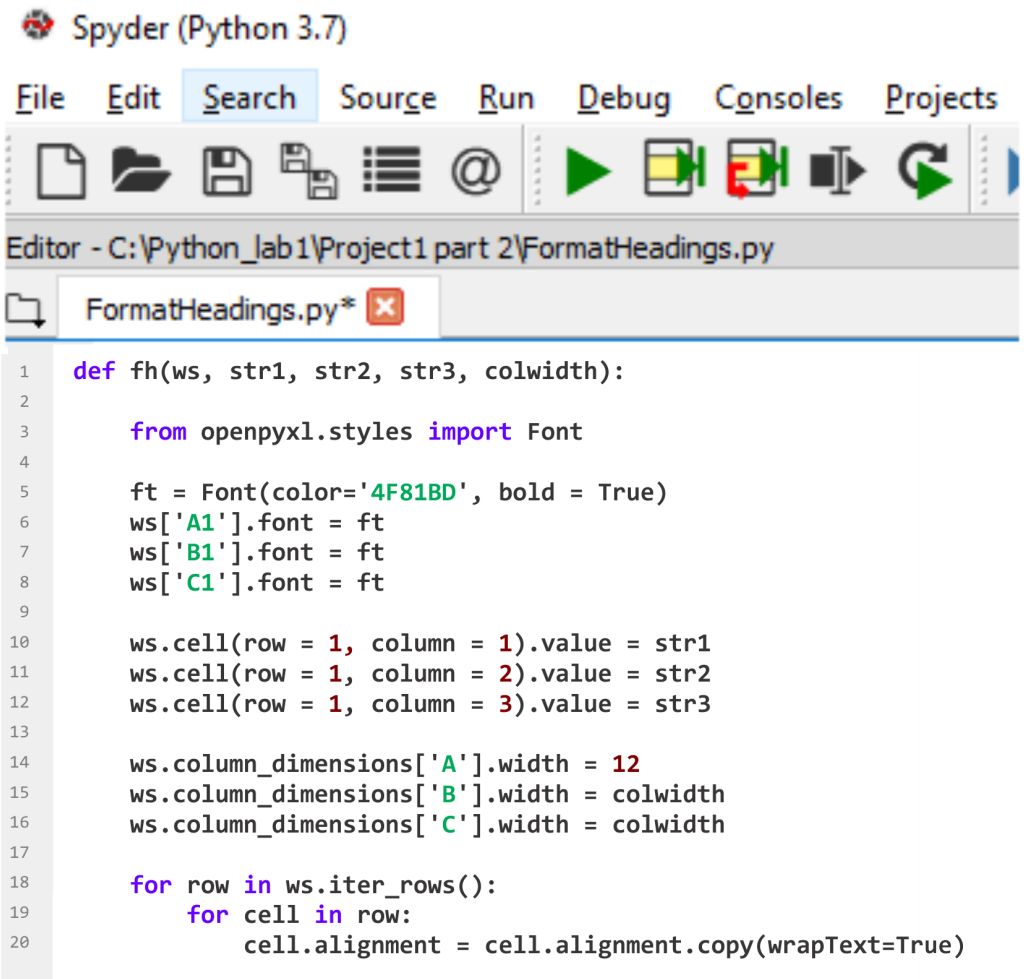
Argument	Value	Description
1	ws4	worksheet object for Diff worksheet
2	Item	Heading Column A
3	Description	Heading Column B
4	Comments	Heading Column C
5	20	Width of Column

Table 4.1 Five Arguments

Line 3 imports the styles method so we can work with the **Font** method.

Lines 5 to 8 set the bold font and blue color for the heading cells. After the program runs, the "Diff.xlsx" file has color headings and the format shown below.

Lines 10 to 12 set the heading name to the values passed in the arguments **str1**, **str2**, and **str3**. Lines 14 to 16 set the column width to the fifth argument **colwidth**. Lines 18 to 20 set cell alignment to wrap text.




```
1   def fh(ws, str1, str2, str3, colwidth):
2
3       from openpyxl.styles import Font
4
5       ft = Font(color='4F81BD', bold = True)
6       ws['A1'].font = ft
7       ws['B1'].font = ft
8       ws['C1'].font = ft
9
10      ws.cell(row = 1, column = 1).value = str1
11      ws.cell(row = 1, column = 2).value = str2
12      ws.cell(row = 1, column = 3).value = str3
13
14      ws.column_dimensions['A'].width = 12
15      ws.column_dimensions['B'].width = colwidth
16      ws.column_dimensions['C'].width = colwidth
17
18      for row in ws.iter_rows():
19          for cell in row:
20              cell.alignment = cell.alignment.copy(wrapText=True)
```

Figure 4.6 FormatHeadings.py

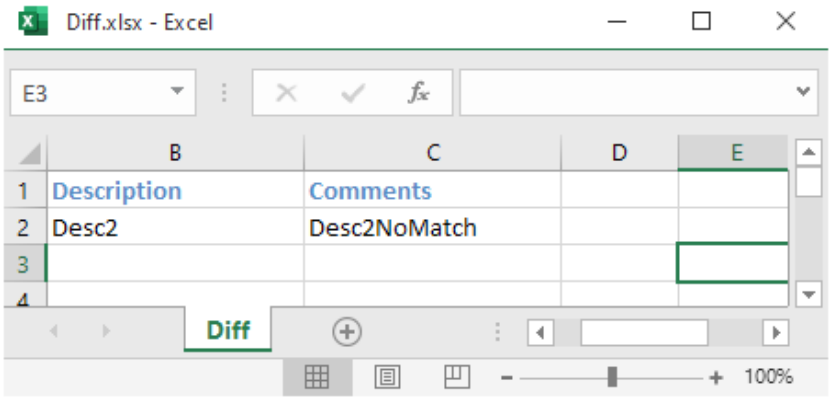


Figure 4.7 Diff.xlsx with Headings

4.6 Delete Existing File

If the output file "Diff.xlsx" already exists, I want to delete it before creating a new file. In "**Project1.1.py**" on line 62 to 64 I import my own file "**IfExistsDelFile.py**" with a function "ifexistsdelete."

```
Line 62: from IfExistsDelFile import ifexistsdelete
Line 63: filepath = diffile
Line 64: ifexistsdelete(filepath)
```

In this example using "**IfExistsDelFile.py**," I import the "**os**" library to work with the Windows file system. The "os.remove" method deletes the file if it exists.

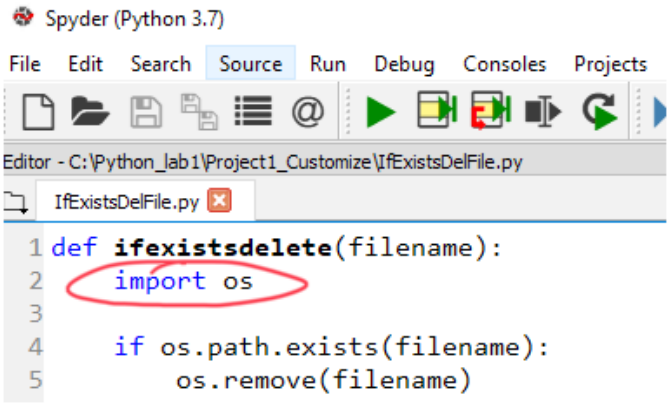


Figure 4.8 IfExistsDelFile.py

4.7 Adding Your Filename, Worksheet, and Column

In the example below, the red arrows highlight lines in the **Project1_Part2_3.py** file. To customize the code to work with your file, worksheet, or column, update the lines shown below.

1. Replace the filename in green on line 3 with your filename.

2. Update line 9 with your worksheet name.

3. Change the column number as appropriate. For example, to use Column B, change the column number in line 14 to "**2**."

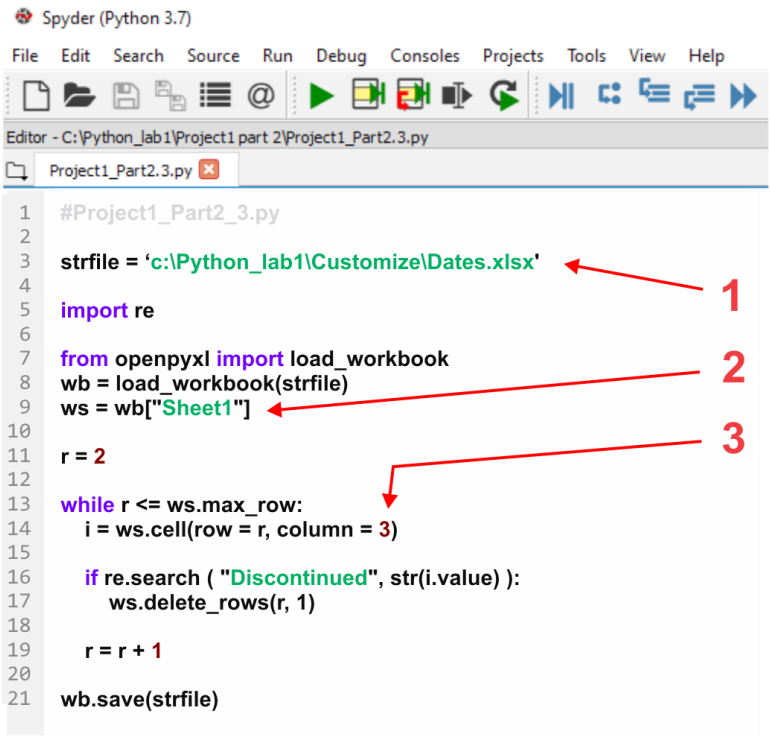


Figure 4.9 Your Filename, Worksheet, and Column

4.8 Item Retired

Did you wonder about the variable "**itemretired**" on line 33? In the **Project1_Part2.6.py** file, on lines 53 to 58, I use "**itemretired**" to mark items in "*Before.xlsx*" that were not found in "*After.xlsx*."

```
29
30 while bfr <= bfrmaxrow:
31     bfritem = ws1.cell(row = bfr, column = 2)
32     aft = 2
33     itemretired = 1
34
35     while aft <= aftmaxrow:
36         aftitem = ws2.cell(row = aft, column = 2)
37         if bfritem.value == aftitem.value:
38             itemretired = 0
39             if ws1.cell(row=bfr, column = 3).value == ws2.cell(row=aft, column = 3).value:
40                 aft = aftmaxrow + 1
41             else: #Column 3 of data is different so write to Diff.xlsx file
42                 ws4.cell(row=dif, column = 1).value = bfritem.value
43                 ws4.cell(row=dif, column = 2).value = ws1.cell(row=bfr, column = 3).value
44                 ws4.cell(row=dif, column = 3).value = ws2.cell(row=aft, column = 3).value
45                 dif = dif + 1
46                 aft = aftmaxrow + 1
47         else:
48             aft = aft + 1 #move to next row in aftfile
49
50     bfr = bfr + 1    #move to next row in bfrfile
51
52 #%%
53     if itemretired == 1:    #the item was removed (not found in the after file)
54         ws5.cell(row = 1, column = 1).value = "Item Name"
55         ws5.cell(row = 1, column = 2).value = "Comments"
56         ws5.cell(row = dif, column = 1).value = bfritem.value
57         ws5.cell(row = dif, column = 2).value = "Item not in After File"
58         dif = dif + 1
59 #%%
```

Figure 4.10 Item Retired

4.9 New Items

In the **Project1_Part2.6.py** file, I added lines 64 to 84 to find new items in "*After.xlsx*" that were not in "*Before.xlsx.*" Notice this code is at the left margin and is **not** part of the previous loop. I also created a new spreadsheet object "ws12" to keep track of new items.

On line 70 I assign 1 to **itemnew.** When an item in the "*After.xlsx*" is not in "*Before.xlsx*," line 74 does not run. When the items match on line 73, line 74 changes **itemnew** to 0.

Line 78 runs after the code loops through all rows in the "*After.xlsx*" file. If **itemnew** still has a value of 1, I update ws12 objects in lines 79 to 82.

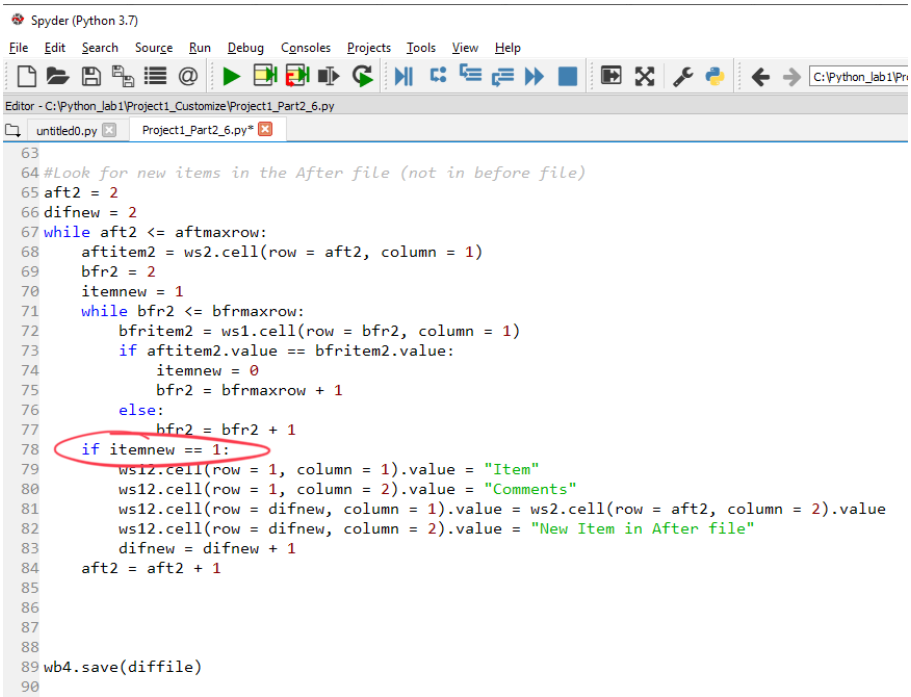


Figure 4.11 New Items

4.10 Compare Dates

The program **Project1_Part2_3.py** uses the "datetime" library to work with dates. This example uses two methods: **datetime** and **strftime**. The **strftime** method converts datetime objects to strings. Line 21 compares the two strings.

- The **datetime** method creates a datetime object in line 14. Line 15 converts the new datetime object to a string.
- The **strftime** takes a datetime object and returns a formatted string representing the date.
- The **strptime** takes a string and returns a datetime object.

Project1_Part2_3.py*

```
1 #Project1_Part2_3.py
2 #Comparing Dates
3
4 bfrfile = 'Before.xlsx'
5
6 from openpyxl import load_workbook
7 wb1 = load_workbook(bfrfile, data_only = True)
8 ws1 = wb1["ExportedData"]
9
10 bfrmaxrow = 7 #total rows with data in bfrfile
11 bfr = 2
12
13 import datetime
14 d1 = datetime.datetime(9999, 12, 31)   #set d1 to 12/31/9999
15 d1 = datetime.datetime.strftime(d1, '%Y-%m-%d' )
16
17 while bfr <= bfrmaxrow:
18     datetochk = ws1.cell(row = bfr, column = 4).value
19     thedate = datetime.datetime.strftime(datetochk,'%Y-%m-%d' )
20     bfritem = ws1.cell(row = bfr, column = 2).value
21     if thedate != d1:
22         print(bfritem, " expired on ", datetochk)
23
24     bfr = bfr + 1
25
```

Figure 4.12 Compare Dates

Another useful datetime function returns today's date.

>>>strtoday = date.**today**()

Conclusion

Einstein said, “If you can’t explain it simply, you don’t understand it well enough.” Learning new things is a passion of mine, and I’ve found the process of organizing notes, creating illustrations, and pondering how to craft clear examples helps me grasp concepts. Then too, it’s nice to go back in a year when I’ve forgotten something and refer to a solid example.

Thank you for reading along with me through the interesting topics and less than thrilling subjects. If the result is you have mastered new features, it was worth it! I’d love to hear the cool things you’re doing with Python, so please don’t hesitate to leave comments in a review.

Appendix

Download the sample Excel files and code from Github. The download includes this manual in PDF form.

https://github.com/cryoung6/Lab1

Please feel free to copy and share any and all materials with others. If you'd be kind enough to leave a review, it might lead to another $1 royalty in my future!

Index