Black Friday Excel Data

```
import openpyxl as xl
from tax_calculator import tax_calculator
from pprint import pprint
```

Part 1

First, read in ski_shop_data.xlsx

Then, the orders worksheet to an intuitive variable name.

```
wb = xl.load_workbook(filename='ski_shop_data.xlsx')

orders = wb['Orders_Info']
```

Column Printer Function

It needs to work with Excel frequently, I will create a helper function to print all the rows a specified column.

This will help to view imported data without looking at Excel, and review the data we've written into columns without needing to save a file to review the changes.

function should take two arguments:

- Worksheet where data is located
- Column Letter of data to print

function should:

- Print Cell Coordinate (e.g. A1, D4).
- Print the contents of the cell.

Then call function twice:

- Once to print Order ID (column A in 'Orders_Info')
- Once to print subtotal (column C in 'Orders_Info')

```
def column_printer(sheet, column):
    for i in range(1, sheet.max_row + 1):
        print(f'{column}{i}', sheet[f'{column}{i}'].value)


def column_printer(sheet, column):
    for cell in sheet[column]:
        print(f'{cell.coordinate}', sheet[cell.coordinate].value)
```

Order Data Dictionary

Create a dictionary with all of the information contained in the 'Orders_Info' Worksheet.

- The dictionary keys should be Order IDs (Column A)
- The values should a list storing the data in the rest of the columns.
- The columns in the list stored as values should be B, C, D, G, and H (after converting to list).
- For example: The first order, column H should be the nested list: [10001, 10002]

A few notes:

- Convert the Items_Ordered field into a list (we covered a helpful string method for this)
- This can be done with a dictionary comprehension
- DO NOT INCLUDE EXCEL COLUMN HEADERS

```
order_dict = {
    orders[f'A{order}'].value:[
        orders[f'B{order}'].value,
        orders[f'C{order}'].value,
        orders[f'D{order}'].value,
        orders[f'G{order}'].value,
        str(orders[f'H{order}'].value).split(', ')
    ]
    for order in range(2, orders.max_row + 1)
}
```

Sales Tax Calculation (Last time!)

It needs to calculate the sales tax and total amount owed for every order in this sheet.

- If location is Sun Valley, apply a sales tax of 8%
- If location is Mammoth, apply a sales tax of 7.75%.
- If location is Stowe, apply a sales tax of 6%

Use the tax_calculator function to apply sales tax to each subtotal.

Insert the calculated sales tax and total amounts into your customer dictionary.

Tip: Figure out how to do this once and then loop through the column.

```
from tax_calculator import tax_calculator


for order in order_dict.values():
    if order[3] == 'Sun Valley':
        transaction = tax_calculator(order[2], .08)
    elif order[3] == 'Mammoth':
        transaction = tax_calculator(order[2], .0775)
    else:
        transaction = tax_calculator(order[2], .06)
    order.insert(3, transaction[1])
    order.insert(4, transaction[2])

pprint(order_dict)

# Ans:


{100000: ['C00004',
          '11/26/2021',
          15.98,
          1.28,
          17.26,
          'Sun Valley',
          ['10001', '10002']],
 100001: ['C00007',
          '11/26/2021',
          899.97,
          54.0,
          953.97,
          'Stowe',
          ['10008', '10009', '10010']],
 100002: ['C00015',
          '11/26/2021',
          799.97,
          62.0,
          861.97,
          'Mammoth',
          ['10011', '10012', '10013']],
 100003: ['C00016',
          '11/26/2021',
          117.96,
          7.08,
          125.04,
          'Stowe',
          ['10002', '10003', '10004', '10006']],
 100004: ['C00020', '11/26/2021', 5.99, 0.48, 6.47, 'Sun Valley',
['10001']],
 100005: ['C00010', '11/26/2021', 599.99, 46.5, 646.49, 'Mammoth',
['10010']],
 100006: ['C00006', '11/26/2021', 24.99, 1.94, 26.93, 'Mammoth',
['10004']],
 100007: ['C00001',
          '11/26/2021',
          1799.94,
          139.5,
          1939.44,
          'Mammoth',
          ['10008', '10008', '10009', '10009', '10009', '10010',
'10010']],
 100008: ['C00003', '11/26/2021', 99.99, 8.0, 107.99, 'Sun Valley',
['10005']],
 100009: ['C00014',
          '11/26/2021',
          254.95,
          20.4,
          275.35,
          'Sun Valley',
          ['10002', '10003', '10004', '10006', '10007']],
 100010: ['C00001',
          '11/26/2021',
          29.98,
          2.32,
          32.3,
          'Mammoth',
          ['10002', '10003']],
 100011: ['C00001', '11/26/2021', 99.99, 7.75, 107.74, 'Mammoth',
['10005']],
 100012: ['C00005',
          '11/26/2021',
          25.98,
          2.08,
          28.06,
          'Sun Valley',
          ['10001', '10003']],
 100013: ['C00008',
          '11/26/2021',
          649.98,
          39.0,
          688.98,
          'Stowe',
          ['10012', '10013']],
 100014: ['C00013', '11/26/2021', 89.99, 7.2, 97.19, 'Sun Valley',
['10014']],
 100020: ['C00004', '11/27/2021', 119.99, 9.6, 129.59, 'Sun Valley',
['10007']],
 100021: ['C00017', '11/27/2021', 599.99, 36.0, 635.99, 'Stowe',
['10010']],
 100022: ['C00019',
          '11/27/2021',
          649.98,
          52.0,
          701.98,
          'Sun Valley',
          ['10012', '10013']],
 100023: ['C00002', '11/27/2021', 24.99, 1.5, 26.49, 'Stowe',
['10004']],
 100024: ['C00008', '11/27/2021', 99.99, 6.0, 105.99, 'Stowe',
['10005']],
 100025: ['C00021', '11/27/2021', 99.99, 7.75, 107.74, 'Mammoth',
['10008']],
 100026: ['C00022', '11/27/2021', 5.99, 0.48, 6.47, 'Sun Valley',
['10001']],
 100027: ['C00006', '11/28/2021', 24.99, 1.94, 26.93, 'Mammoth',
['10002']],
 100031: ['C00018',
          '11/28/2021',
          999.96,
          60.0,
          1059.96,
          'Stowe',
          ['10005', '10008', '10009', '10010']],
 100032: ['C00018', '11/28/2021', 99.99, 6.0, 105.99, 'Stowe',
['10006']],
 100033: ['C00010',
          '11/28/2021',
          399.97,
          31.0,
          430.97,
          'Mammoth',
          ['10005', '10008', '10009']],
 100034: ['C00016', '11/28/2021', 89.99, 5.4, 95.39, 'Stowe',
['10014']]}
```

Write Sales Tax and Total Into the Excel Sheet

Great job! Now its just needed to write this data into the workbook.

Write the sales tax and total you just calculated into the workbook, then save!

Call this workbook 'ski_shop_data_fixed'.

Tip: There are a few ways to do this. As always, be patient, solve one step at a time.

```python
for index, order in enumerate(order_dict.values(), start=2):
    orders[f'E{index}'] = order[3]
    orders[f'F{index}'] = order[4]

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

Part 2

Analysis Time!

Now that it has fixed the data - it's time to perform analysis on our sales.

The starting point will be the order_dict we created, after we added the taxes and totals columns.

The first step will be to write a function that calculates the sum of a 'column' of data in our dictionary.

A 'column' for example, would be subtotals, which is at index 2 in the list stored as our dictionary values.

Function should take the following arguments:

- Column Index
- Dictionary Name

It should output:

- The sum of values in the column (rounded to two decimal places)

Assume only numeric values will be in the column (You can develop cleaning logic later :D)

Tip: Use a list comprehension to retrieve the values of interest.

```
def column_sum(column_index, dictionary):
    return round(sum([value[column_index] for value in 
dictionary.values()]), 2)
```

Sum The Subtotal, Tax, and Total Columns

Now that there is column sum function, calculate the sum of:

- Subtotals
- Taxes
- Totals

```
# column index variable names
print(column_sum(2, order_dict))
print(column_sum(3, order_dict))
print(column_sum(4, order_dict))

# Ans:


8731.47
617.2
9348.67
```

What is the average of our subtotals?

Calculate the average value of transactions.

Remember that each entry in dictionary is one order.

```
```

```python
round(column_sum(2, order_dict) / len(order_dict), 2)

# Ans:

323.39
```

```python
subtotals = [value[2] for value in order_dict.values()]

round(sum(subtotals) / len(subtotals), 2)

# Ans:

323.39
```

How many unique customers did we have?

Calculate the total number of unique customers in our sales data.

Then calculate the number of orders per customer (total orders/ unique customers).

```python
unique_customers = len(set([value[0] for value in
order_dict.values()]))

orders_per_customer = len(order_dict) / unique_customers

print(unique_customers, orders_per_customer)


# Ans:

19 1.4210526315789473
```

How many items in total did we sell?

Calculate the total number of items we sold in across all orders.

This information is in Column H, which should be the last element in order_dict's values.

```python
sum([len(value[6]) for value in order_dict.values()])


# Ans:

54
```

Sales By Location

Calculate the sum of subtotals by location.

Create a dictionary to store them, where location is the key, and revenue for that location is the value.

A few steps to consider:

- Loop through you dictionary
- build a dictionary as you go with location as key
- increment revenue every time a transaction matches the location.

Output should look like {'Location1': sum of subtotals for 'Location1'}

With an entry for each location.

```
location_sums = {}

for data in order_dict.values():

    location = data[5]


    if location not in location_sums:

        location_sums[location] = 0

    location_sums[location] += data[2]

location_sums

# Ans:


{'Sun Valley': 1268.84,
 'Stowe': 3582.8199999999993,
 'Mammoth': 3879.8099999999995}
```

Challenge: Aggregator Function

Now that revenue by category have been summed, can you write a function to generalize calculating a sum of a column, grouped by the unique values in another column? (for example, sum of totals by date or customer_id).

Your function should take the following arguments:

- index of the 'column' (index position in order_dict) to group by
- index of the 'column' (index position in order_dict) to sum by category
- the dictionary where the data is located (assume the same structure as order_dict.

It should return:

- A dictionary with the categories as keys, and the sum by category as value.

Once you've done so use your function to sum totals by date and customer_id.

```python
def aggregator(category_index, field_to_sum_index, dictionary):
    category_sums = {}

    for data in dictionary.values():

        category = data[category_index]

        if category not in category_sums:
            category_sums[category] = 0
        category_sums[category] += data[field_to_sum_index]
    return category_sums
```

For fun, if you got tired of looking up indices, you could assign your indices names. In libraries like Pandas it'll be able to reference DataFrame column names or indices.

Here unpacking a tuple will be done to name our column indices

```python
customer_id, date, subtotals, taxes, totals, location, items_sold =
(0, 1, 2, 3, 4, 5, 6)

# Function call with numeric indices
aggregator(1, 2, order_dict)

# Ans:


{'11/26/2021': 5515.649999999998, '11/27/2021': 1600.92, '11/28/2021':
1614.9}



# Function call with 'named' indices
aggregator(customer_id, totals, order_dict)

# Ans:


{'C00004': 146.85,
 'C00007': 953.97,
 'C00015': 861.97,
 'C00016': 220.43,
 'C00020': 6.47,
 'C00010': 1077.46,
 'C00006': 53.86,
 'C00001': 2079.48,
 'C00003': 107.99,
 'C00014': 275.35,
 'C00005': 28.06,
 'C00008': 794.97,
 'C00013': 97.19,
 'C00017': 635.99,
 'C00019': 701.98,
 'C00002': 26.49,
 'C00021': 107.74,
 'C00022': 6.47,
 'C00018': 1165.95}
```