**Challenge 3 – Brain Dump/Notes**

*More notes on physical notebook*

**Help adding to GitHub – original empty files**

To add a folder to your local Git repository, you can follow these steps using Git Bash:

1. **Navigate to Your Local Repository**: Open Git Bash and navigate to the directory of your local Git repository where you want to add the new folder. You can use the cd command to change directories.
2. **Create a New Folder**: Use the mkdir command to create a new folder. For example, to create a folder named NewFolder, you can run:
3. mkdir NewFolder
4. **Check the Status**: Use the git status command to see the status of your repository. It will show you the untracked new folder.
5. **Add the New Folder**: To stage the new folder for the commit, use the git add command. For example, to add the NewFolder, run:
6. git add NewFolder
7. **Commit the Changes**: After adding the new folder, commit the changes to your local repository with a descriptive message using the git commit command. For example:
8. git commit -m "Add NewFolder to the repository"
9. **Verify the Changes**: You can use git status again to verify that the new folder has been added and committed.

By following these steps, you can add a new folder to your local Git repository using Git Bash.

**PyBank**

**Help remembering how to loop through column to find unique months**

Here's an example of how you can loop through a column in a CSV file containing dates in the format "10-Jan" to find the number of unique months:

import csv

# Open the CSV file

with open('data.csv', 'r') as file:

reader = csv.DictReader(file)

# Create a set to store unique months

unique\_months = set()

# Loop through the rows in the CSV file

for row in reader:

date = row['date\_column']

month = date.split("-")[1]

unique\_months.add(month)

# Count the number of unique months

num\_unique\_months = len(unique\_months)

print("Number of unique months:", num\_unique\_months)

In this code snippet, we use the csv module to read the CSV file, loop through the rows, extract the month from the specified column, and store unique months in a set. Finally, we calculate the number of unique months in the column. You can run this code in VSCode to get the desired output without using Pandas.

***Had issues with path***

1. **Verify the File Structure**: Ensure that the folder structure of your project matches the path you are trying to create. Check if the budget\_data.csv file is located in the correct directory relative to your Python script.

***Code for total results in 0. I did something wrong…***

**It seems like the issue in your code is related to how you are reading the CSV file and looping through it. After looping through the file once to count the number of unique months, the file pointer is at the end of the file. When you try to loop through the file again to calculate the total profit/loss, there are no more rows left to read, resulting in a total of 0.**

To address this issue, you can either read the file again from the beginning or store the data you need in memory after the first loop. Here's an updated version of your code that reads the file once and calculates both the total number of months and the net total profit/loss correctly:

import os

import csv

bankpath = os.path.join('..', 'Resources', 'Budget\_data.csv')

# Initialize variables

unique\_months = set()

total\_profit\_losses = 0

# Read the CSV file

with open(bankpath, encoding="utf-8") as bankfile:

bankreader = csv.reader(bankfile)

next(bankreader) # Skip the header row

for row in bankreader:

date = row[0]

month = date.split("-")[0] # Assuming the date format is "MMM-YY"

unique\_months.add(month)

profit\_loss = int(row[1]) # Convert profit/loss to integer

total\_profit\_losses += profit\_loss

# Calculate total number of months

num\_months = len(unique\_months)

print("Total Months:", num\_months)

# Calculate net total Profit/Loss

print("Total: $", total\_profit\_losses)

In this updated code:

* I combined the loops to read the file only once.
* I converted the profit/loss values to integers before summing them up to get the total profit/loss.
* I used the correct format for splitting the date to extract the month.

***Tired to consolidate code to not have issues when trying to do total. Having some difficulties merging above recommendations with my original code idea for the sum***

I see you're trying to calculate the total profit/losses from a CSV file. There are a few issues in your code that need to be addressed:

1. When you read the CSV file, you should skip the header row.
2. The index for the Profit/Losses column is usually 1, not 0.
3. You are trying to convert the profit/loss value to an integer and then split it, which is not necessary for this calculation.

Here is the corrected code snippet for calculating the total profit/losses:

import os

import csv

bankpath = os.path.join('..', 'Resources', 'Budget\_data.csv')

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

with open(bankpath, encoding="utf-8") as bankfile:

bankreader = csv.reader(bankfile)

next(bankreader) # Skip the header row

for row in bankreader:

date = row[0] # Assuming the date is in the first column

month = date.split("-")[0]

unique\_months.add(month)

profit\_loss = int(row[1]) # Assuming profit/loss is in the second column

TOTAL\_PROFIT\_LOSSES += profit\_loss

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

print("Total: $", TOTAL\_PROFIT\_LOSSES)

Please replace row[0] and row[1] with the correct indices based on the structure of your CSV file. Additionally, ensure that the profit/loss values in the CSV file are in a format that can be directly converted to an integer.

***Get error when try to use above recc. Was because forgot to skip headers again (had done so before a bunch of changes). Fixed and now working!!***

# get file and using modules for it

import os

import csv

bankpath = os.path.join('..','Resources','Budget\_data.csv')

#initialize starters

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

# Reading using csv module. Open CSV file

with open(bankpath, encoding="utf-8") as bankfile:

    bankreader = csv.reader(bankfile)

    next(bankreader)  # Skip the header row

    for row in bankreader:

        date = row[0]  # the date is in the first column

        month = date.split(",")[0]

        unique\_months.add(month)

        profit\_loss = int(row[1])  # Assuming profit/loss is in the second column

        TOTAL\_PROFIT\_LOSSES += profit\_loss

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

# Calculate net total Profit/Loss

print("Total: $", TOTAL\_PROFIT\_LOSSES)

***difficulty understanding how to track changes in profit/loss. XpertLearning gave following suggestion***

* We create a list changes\_in\_profit\_losses to store the changes in profit/losses over the entire period.
* We iterate through the profit/loss values and calculate the change between the current profit/loss and the previous profit/loss.
* We calculate the average change by summing all changes and dividing by the total number of changes.
* Finally, we print the changes in profit/losses and the average change rounded to two decimal places.

***Xpert suggestion for code snippet***

import os

import csv

bankpath = os.path.join('..', 'Resources', 'Budget\_data.csv')

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

with open(bankpath, encoding="utf-8") as bankfile:

bankreader = csv.reader(bankfile)

next(bankreader) # Skip the header row

for row in bankreader:

profit\_loss = int(row[1]) # Assuming profit/loss is in the second column

if previous\_profit\_loss is not None:

change = profit\_loss - previous\_profit\_loss

changes\_in\_profit\_losses.append(change)

previous\_profit\_loss = profit\_loss

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

print("Changes in Profit/Losses over the entire period:")

print(changes\_in\_profit\_losses)

print("Average Change: $", round(average\_change, 2))

***I adjusted things and it works for the avg change!!***

# get file and using modules for it

import os

import csv

bankpath = os.path.join('..','Resources','Budget\_data.csv')

#initialize starters

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

# Reading using csv module. Open CSV file

with open(bankpath, encoding="utf-8") as bankfile:

    bankreader = csv.reader(bankfile)

    next(bankreader)  # Skip the header row

    for row in bankreader:

        date = row[0]  # the date is in the first column

        month = date.split(",")[0]

        unique\_months.add(month)

        profit\_loss = int(row[1])  # Assuming profit/loss is in the second column

        TOTAL\_PROFIT\_LOSSES += profit\_loss

        #if conditional to see changes in current vs previous row

        if previous\_profit\_loss is not None: #[]

            change = profit\_loss - previous\_profit\_loss

            changes\_in\_profit\_losses.append(change)

        previous\_profit\_loss = profit\_loss #for next comparison, make current the "previous"

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

# Calculate net total Profit/Loss

print("Total: $", TOTAL\_PROFIT\_LOSSES)

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

print("Average Change: $", round(average\_change, 2))

***added in calcs for greatest incr and decr and values correct BUT dates keep changing every time I run it***

# get file and using modules for it

import os

import csv

bankpath = os.path.join('..','Resources','Budget\_data.csv')

#initialize starters

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

# Reading using csv module. Open CSV file

with open(bankpath, encoding="utf-8") as bankfile:

    bankreader = csv.reader(bankfile)

    next(bankreader)  # Skip the header row

    for row in bankreader:

        date = row[0]  # the date is in the first column

        month = date.split(",")[0]

        unique\_months.add(month)

        profit\_loss = int(row[1])  # Assuming profit/loss is in the second column

        TOTAL\_PROFIT\_LOSSES += profit\_loss

        #if conditional to see changes in current vs previous row

        if previous\_profit\_loss is not None: #[]

            change = profit\_loss - previous\_profit\_loss

            changes\_in\_profit\_losses.append(change) #add each change from each comparison to list with changes

        previous\_profit\_loss = profit\_loss #for next comparison, make current the "previous"

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

# Calculate net total Profit/Loss

print("Total: $", TOTAL\_PROFIT\_LOSSES)

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

print("Average Change: $", round(average\_change, 2))

#now that have list of the changes over the period, can just do min and max for greatest incr and decr.placing here so after loop ends

greatest\_increase = max(changes\_in\_profit\_losses)

greatest\_decrease = min(changes\_in\_profit\_losses) #hopefully works keeping into account negative values

# Find the corresponding dates for the greatest increase and decrease. using index function

index\_greatest\_increase = changes\_in\_profit\_losses.index(greatest\_increase)

index\_greatest\_decrease = changes\_in\_profit\_losses.index(greatest\_decrease)

#make list by converting unique months set. maybe should have made it a list from the beginning?

dates = list(unique\_months)

date\_greatest\_increase = dates[index\_greatest\_increase+1]  # Add 1 to account for skipping the first row

date\_greatest\_decrease = dates[index\_greatest\_decrease+1]  # Add 1 to account for skipping the first row

print("Greatest Increase in Profits:", date\_greatest\_increase, "($", greatest\_increase, ")")

print("Greatest Decrease in Profits:", date\_greatest\_decrease, "($", greatest\_decrease, ")")

#dates are different when I run them? but min and mx number stays the same...

***Did rest but having difficulty outputting to new txt file. The following prints out Financial Analysis as indiv letters in file and the list with the title of the values but not the name and values themselves…***

# get file and using modules for it

import os

import csv

bankpath = os.path.join('..','Resources','Budget\_data.csv')

#initialize starters

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

# Reading using csv module. Open CSV file

with open(bankpath, encoding="utf-8") as bankfile:

    bankreader = csv.reader(bankfile)

    next(bankreader)  # Skip the header row

    for row in bankreader:

        date = row[0]  # the date is in the first column

        month = date.split(",")[0]

        unique\_months.add(month)

        profit\_loss = int(row[1])  # Assuming profit/loss is in the second column

        TOTAL\_PROFIT\_LOSSES += profit\_loss

        #if conditional to see changes in current vs previous row

        if previous\_profit\_loss is not None: #[]

            change = profit\_loss - previous\_profit\_loss

            changes\_in\_profit\_losses.append(change) #add each change from each comparison to list with changes

        previous\_profit\_loss = profit\_loss #for next comparison, make current the "previous"

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

Results\_Months= print("Total Months:", NUM\_MONTHS)

# Calculate net total Profit/Loss

print("Total: $", TOTAL\_PROFIT\_LOSSES)

Results\_Total = print("Total: $", TOTAL\_PROFIT\_LOSSES)

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

print("Average Change: $", round(average\_change, 2))

Results\_Avg\_Change = print("Average Change: $", round(average\_change, 2))

#now that have list of the changes over the period, can just do min and max for greatest incr and decr.placing here so after loop ends

greatest\_increase = max(changes\_in\_profit\_losses)

greatest\_decrease = min(changes\_in\_profit\_losses) #hopefully works keeping into account negative values

# Find the corresponding dates for the greatest increase and decrease. using index function to get row/placement

index\_greatest\_increase = changes\_in\_profit\_losses.index(greatest\_increase)

index\_greatest\_decrease = changes\_in\_profit\_losses.index(greatest\_decrease)

#make list by converting unique months set. maybe should have made it a list from the beginning?

dates = list(unique\_months)

date\_greatest\_increase = dates[index\_greatest\_increase+1]  # Add 1 to account for skipping the first row

date\_greatest\_decrease = dates[index\_greatest\_decrease+1]  # Add 1 to account for skipping the first row

print("Greatest Increase in Profits:", date\_greatest\_increase, "($", greatest\_increase, ")")

Results\_Gr\_Inc= print("Greatest Increase in Profits:", date\_greatest\_increase, "($", greatest\_increase, ")")

print("Greatest Decrease in Profits:", date\_greatest\_decrease, "($", greatest\_decrease, ")")

Results\_Gr\_Decr=print("Greatest Decrease in Profits:", date\_greatest\_decrease, "($", greatest\_decrease, ")")

#dates are different when I run them? but min and mx number stays the same...

# Specify the file to write to

output\_path = os.path.join('..', 'Analysis', 'PyBank\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

with open(output\_path, 'w') as csvfile:

    # Initialize csv.writer

    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)

    csvwriter.writerow('Financial Analysis')

    # Write the second row

    csvwriter.writerow(['Results\_Months','Results\_Total','Results\_Avg\_Change','Results\_Gr\_Inc','Results\_Gr\_Decr'])

***help exporting results***

**To export the results of your code to a text file, you can follow these steps:**

1. Open a new file in write mode using open() function and provide a file path where you want to save the results.
2. Write the results to the file using the write() method.
3. Close the file to save the changes.

Here is how you can modify your code to export the results to a text file:

import os

import csv

# Define the file paths

bankpath = os.path.join('..', 'Resources', 'Budget\_data.csv')

output\_file = os.path.join('output.txt')

# Initialize variables

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

# Reading the CSV file

with open(bankpath, encoding="utf-8") as bankfile:

bankreader = csv.reader(bankfile)

next(bankreader) # Skip the header row

for row in bankreader:

date = row[0].split(",")[0]

unique\_months.add(date)

profit\_loss = int(row[1])

TOTAL\_PROFIT\_LOSSES += profit\_loss

if previous\_profit\_loss is not None:

change = profit\_loss - previous\_profit\_loss

changes\_in\_profit\_losses.append(change)

previous\_profit\_loss = profit\_loss

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

# Calculate net total Profit/Loss

TOTAL\_PROFIT\_LOSSES\_STR = "${:,.2f}".format(TOTAL\_PROFIT\_LOSSES)

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

# Find the greatest increase and decrease

greatest\_increase = max(changes\_in\_profit\_losses)

greatest\_decrease = min(changes\_in\_profit\_losses)

# Find the corresponding dates for the greatest increase and decrease

index\_greatest\_increase = changes\_in\_profit\_losses.index(greatest\_increase)

index\_greatest\_decrease = changes\_in\_profit\_losses.index(greatest\_decrease)

dates = list(unique\_months)

date\_greatest\_increase = dates[index\_greatest\_increase+1]

date\_greatest\_decrease = dates[index\_greatest\_decrease+1]

# Print the results

print("Financial Analysis")

print("----------------------------")

print(f"Total Months: {NUM\_MONTHS}")

print(f"Total: {TOTAL\_PROFIT\_LOSSES\_STR}")

print(f"Average Change: ${round(average\_change, 2)}")

print(f"Greatest Increase in Profits: {date\_greatest\_increase} (${greatest\_increase})")

print(f"Greatest Decrease in Profits: {date\_greatest\_decrease} (${greatest\_decrease})")

# Export the results to a text file

with open(output\_file, 'w') as file:

file.write("Financial Analysis\n")

file.write("----------------------------\n")

file.write(f"Total Months: {NUM\_MONTHS}\n")

file.write(f"Total: {TOTAL\_PROFIT\_LOSSES\_STR}\n")

file.write(f"Average Change: ${round(average\_change, 2)}\n")

file.write(f"Greatest Increase in Profits: {date\_greatest\_increase} (${greatest\_increase})\n")

file.write(f"Greatest Decrease in Profits: {date\_greatest\_decrease} (${greatest\_decrease})\n")

print("Results exported to output.txt file.")

This code will export the financial analysis results to a text file named output.txt.

***Adjusted code to print correctly using Xpert Learning. See end below***

# get file and using modules for it

import os

import csv

bankpath = os.path.join('..','Resources','Budget\_data.csv')

#initialize starters

unique\_months = set()

TOTAL\_PROFIT\_LOSSES = 0

changes\_in\_profit\_losses = []

previous\_profit\_loss = None

# Reading using csv module. Open CSV file

with open(bankpath, encoding="utf-8") as bankfile:

    bankreader = csv.reader(bankfile)

    next(bankreader)  # Skip the header row

    for row in bankreader:

        date = row[0]  # the date is in the first column

        month = date.split(",")[0]

        unique\_months.add(month)

        profit\_loss = int(row[1])  # Assuming profit/loss is in the second column

        TOTAL\_PROFIT\_LOSSES += profit\_loss

        #if conditional to see changes in current vs previous row

        if previous\_profit\_loss is not None: #[]

            change = profit\_loss - previous\_profit\_loss

            changes\_in\_profit\_losses.append(change) #add each change from each comparison to list with changes

        previous\_profit\_loss = profit\_loss #for next comparison, make current the "previous"

# Calculate total number of months

NUM\_MONTHS = len(unique\_months)

print("Total Months:", NUM\_MONTHS)

# Calculate net total Profit/Loss

print("Total: $", TOTAL\_PROFIT\_LOSSES)

# Calculate the average of changes in Profit/Losses

average\_change = sum(changes\_in\_profit\_losses) / len(changes\_in\_profit\_losses)

print("Average Change: $", round(average\_change, 2))

#now that have list of the changes over the period, can just do min and max for greatest incr and decr.placing here so after loop ends

greatest\_increase = max(changes\_in\_profit\_losses)

greatest\_decrease = min(changes\_in\_profit\_losses) #hopefully works keeping into account negative values

# Find the corresponding dates for the greatest increase and decrease. using index function to get row/placement

index\_greatest\_increase = changes\_in\_profit\_losses.index(greatest\_increase)

index\_greatest\_decrease = changes\_in\_profit\_losses.index(greatest\_decrease)

#make list by converting unique months set. maybe should have made it a list from the beginning?

dates = list(unique\_months)

date\_greatest\_increase = dates[index\_greatest\_increase+1]  # Add 1 to account for skipping the first row

date\_greatest\_decrease = dates[index\_greatest\_decrease+1]  # Add 1 to account for skipping the first row

print("Greatest Increase in Profits:", date\_greatest\_increase, "($", greatest\_increase, ")")

print("Greatest Decrease in Profits:", date\_greatest\_decrease, "($", greatest\_decrease, ")")

#dates are different when I run them? but min and mx number stays the same...

# Specify the file to write to

output\_path = os.path.join('..','Analysis','PyBank\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

with open(output\_path, 'w') as file:

    file.write("Financial Analysis\n")

    file.write("----------------------------\n")

    file.write(f"Total Months: {NUM\_MONTHS}\n")

    file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    file.write(f"Average Change: ${round(average\_change, 2)}\n")

    file.write(f"Greatest Increase in Profits: {date\_greatest\_increase} (${greatest\_increase})\n")

    file.write(f"Greatest Decrease in Profits: {date\_greatest\_decrease} (${greatest\_decrease})\n")

print("Results exported to PyBank\_Results.txt")

**PyPoll**

***Chose list instead of set to initialize vote counter***

**difference between set and list python vscode**

In Python, both sets and lists are data structures used to store multiple elements. Here are some key differences between sets and lists in Python:

1. **Ordering:**
   * Lists: Lists are ordered collections, meaning the elements in a list are stored in a specific order and can be accessed by their index.
   * Sets: Sets are unordered collections, meaning the elements in a set do not have a specific order.
2. **Duplicates:**
   * Lists: Lists can contain duplicate elements. Each element in a list can appear multiple times.
   * Sets: Sets do not allow duplicate elements. If you try to add a duplicate element to a set, it will be ignored.
3. **Mutability:**
   * Lists: Lists are mutable, which means you can change, add, or remove elements from a list after it has been created.
   * Sets: Sets are mutable as well, but the elements inside a set are immutable. You can add or remove elements from a set, but you cannot change the elements themselves.
4. **Accessing Elements:**
   * Lists: Elements in a list are accessed by their index, using square brackets [].
   * Sets: Elements in a set are not accessed by index since sets are unordered. You typically check for the presence of an element in a set rather than accessing it by index.
5. **Performance:**
   * Sets are optimized for fast membership testing (checking if an element is in the set), while lists are more versatile for general-purpose storage and manipulation of elements.

In summary, if you need to store elements in a specific order, allow duplicates, and need to access elements by index, you should use a list. If you need to store unique elements, perform set operations like union, intersection, and difference, and need fast membership testing, then a set would be more appropriate.

***Had to make a set. Will later change to list. Bc list made error come up bc not inputting index number in []. Hopefully does not change result for a variable like did in PyBank***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters

unique\_votes=set()

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to list after each loop

#calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)

***I started to confuse myself for whatever reason so I reassured myself about what row[0] means***  
**in csv file in python vscode script. does row[0] look at 1st row or 1st column**

In Python, when you access row[0] in a CSV file, it refers to the first column of the current row being processed, not the first row. Each row in a CSV file is typically represented as a list of values, where each element in the list corresponds to a column in that row.

So, row[0] would give you the value in the first column of the current row, row[1] would give you the value in the second column, and so on. If you want to access the first row in the CSV file, you would need to read the file and then access the elements in the list representing that row.

***Did unique candidate set to use for later calcs. Checked I had correct num unique candidates (3). Worked 😊***

***To check to see if I can print name of each candidate, I tried to print. Could not change sets to list at top or cause errors with adding looped values as elements. Needed to later convert set to list for candidates.***  
**debug # checking to see if can print each candidate. have to change that set to a list. print(unique\_candidates)**

If you are encountering an issue with printing each candidate and need to access elements by index, you should convert the set unique\_candidates to a list. Here's an example of how you can convert a set to a list in Python:

# Convert the set to a list

unique\_candidates\_list = list(unique\_candidates)

# Now you can iterate over the list and print each candidate

for candidate in unique\_candidates\_list:

print(candidate)

By converting the set to a list, you can then iterate over the list and print each candidate. This way, you can access elements by index if needed.

***Got the below but it keeps printing R then C then D (out of alphabetical order)***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters. made an empty set where applicable

unique\_votes= set()

unique\_candidates= set()

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to set after each loop

        candidate\_column=row[2] #3rd column

        candidate = candidate\_column.split(",")[0] #1st elmen after comma split?

        #note to self: I don't think split was necessary but was not sure how to add to set without this

        unique\_candidates.add(candidate) #add unique candidate to set after each loop

#calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

#calc total candidates. do not need to print

NUM\_CANDIDATES=len(unique\_candidates)

#print(NUM\_CANDIDATES)

# checking to see if can print each candidate. cheange set to list

# can fix this later to print each with its vote count?

unique\_candidates\_list = list(unique\_candidates)

# Sort the combined list alphabetically based on the names

candidates\_sorted = sorted(unique\_candidates\_list, key=lambda x: x[1])

for candidate in candidates\_sorted:

    print(candidate)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)

**It seems like there is an issue with sorting the list of unique candidates alphabetically. The error is in the sorting part of the code. When using sorted() with a list of strings, you should sort the list directly without trying to access the second element of each string in the lambda function.**

**Here is the corrected code snippet to sort and print the unique candidates alphabetically:**

import os

import csv

pollpath = os.path.join('..', 'Resources', 'election\_data.csv')

unique\_votes = set()

unique\_candidates = set()

with open(pollpath, encoding="utf-8") as pollfile:

pollreader = csv.reader(pollfile)

next(pollreader) # Skip the header row

for row in pollreader:

ballot\_id\_column = row[0]

vote = ballot\_id\_column.split(",")[0]

unique\_votes.add(vote)

candidate\_column = row[2]

candidate = candidate\_column.split(",")[0]

unique\_candidates.add(candidate)

NUM\_VOTES = len(unique\_votes)

print("Total Votes: ", NUM\_VOTES)

# Convert set to list for sorting

unique\_candidates\_list = list(unique\_candidates)

# Sort the list alphabetically

candidates\_sorted = sorted(unique\_candidates\_list)

for candidate in candidates\_sorted:

print(candidate)

This code snippet should now correctly sort and print the unique candidates alphabetically.

***Above worked! Candidates now in alphabetical order. Use later to print candidates with their respective votes counts?***

***Looked for help counting votes for each candidate in csv. Ended up not needing the work I did above to get them in alphabetical order***   
**To count how many times each candidate appears in the CSV file, you can modify your existing code to keep track of the vote count for each candidate. Here's how you can update your code to achieve this:**

import os

import csv

pollpath = os.path.join('..', 'Resources', 'election\_data.csv')

# Initialize dictionaries to store the vote count for each candidate

candidate\_votes = {}

# Reading the CSV file using the csv module

with open(pollpath, encoding="utf-8") as pollfile:

pollreader = csv.reader(pollfile)

next(pollreader) # Skip the header row

for row in pollreader:

candidate = row[2] # Candidate name is in the 3rd column

if candidate in candidate\_votes:

candidate\_votes[candidate] += 1

else:

candidate\_votes[candidate] = 1

# Calculate total votes

total\_votes = sum(candidate\_votes.values())

print("Total Votes: ", total\_votes)

# Display the vote count for each candidate

for candidate, votes in candidate\_votes.items():

print(f"{candidate}: {votes} votes")

# Determine the winner based on the maximum votes

winner = max(candidate\_votes, key=candidate\_votes.get)

print("Winner: ", winner)

***Below. Got votes for each candidate in alphabetical order***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters. made an empty set where applicable

unique\_votes= set()

unique\_candidates= set()

candidate\_votes = {} #use dictironary to hold vote for each candidate

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to set after each loop

        candidate\_column=row[2] #3rd column

        candidate = candidate\_column.split(",")[0] #1st elmen after comma split?

        #note to self: I don't think split was necessary but was not sure how to add to set without this

        #later saw could do candidate = row[2]

        unique\_candidates.add(candidate) #add unique candidate to set after each loop

        #do loop to count each instance of each unique candidate

        if candidate in candidate\_votes:

            candidate\_votes[candidate] += 1

        else:

            candidate\_votes[candidate] = 1

#calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

#calc total candidates. do not need to print; just checking I did it correctly

NUM\_CANDIDATES=len(unique\_candidates)

#print(NUM\_CANDIDATES)

# checking to see if can print each candidate. cheange set to list

# can fix this later to print each with its vote count?

unique\_candidates\_list = list(unique\_candidates)

# Display the vote count for each candidate

for candidate, votes in candidate\_votes.items():

    print(f"{candidate}: {votes}")

# Sort alphabetically based on the names and print. use this later once calc votes for each above this line

#candidates\_sorted = sorted(unique\_candidates\_list)

#for candidate in candidates\_sorted:

#    print(candidate)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)

***Calc winner worked. Now need percent votes for each candidate.***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters. made an empty set where applicable

unique\_votes= set()

unique\_candidates= set()

candidate\_votes = {} #use dictironary to hold vote for each candidate

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to set after each loop

        candidate\_column=row[2] #3rd column

        candidate = candidate\_column.split(",")[0] #1st elmen after comma split?

        #note to self: I don't think split was necessary but was not sure how to add to set without this

        #later saw could do candidate = row[2]

        unique\_candidates.add(candidate) #add unique candidate to set after each loop

        #do loop to count each instance of each unique candidate

        if candidate in candidate\_votes:

            candidate\_votes[candidate] += 1

        else:

            candidate\_votes[candidate] = 1

#calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

#calc total candidates. do not need to print; just checking I did it correctly

NUM\_CANDIDATES=len(unique\_candidates)

#print(NUM\_CANDIDATES)

# checking to see if can print each candidate. cheange set to list

# can fix this later to print each with its vote count?

unique\_candidates\_list = list(unique\_candidates)

# Display the vote count for each candidate

for candidate, votes in candidate\_votes.items():

    print(f"{candidate}: {votes}")

# Sort alphabetically based on the names and print. use this later once calc votes for each above this line

#candidates\_sorted = sorted(unique\_candidates\_list)

#for candidate in candidates\_sorted:

#    print(candidate)

#Calculate winner based on maximum votes. Use dictonary key ID to ID name of candidate bc using dictionary

winner = max(candidate\_votes, key=candidate\_votes.get)

print("Winner: ", winner)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)

***Calc’d percentage votes for each candidate and worked but decimal points too long.***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters. made an empty set where applicable

unique\_votes= set()

unique\_candidates= set()

candidate\_votes = {} #use dictironary to hold vote for each candidate

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to set after each loop

        candidate\_column=row[2] #3rd column

        candidate = candidate\_column.split(",")[0] #1st elmen after comma split?

        #note to self: I don't think split was necessary but was not sure how to add to set without this

        #later saw could do candidate = row[2]

        unique\_candidates.add(candidate) #add unique candidate to set after each loop

        #do loop to count each instance of each unique candidate

        if candidate in candidate\_votes:

            candidate\_votes[candidate] += 1

        else:

            candidate\_votes[candidate] = 1

        #loop to cal percentage of each

#Calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

#calc total candidates. do not need to print; just checking I did it correctly

#NUM\_CANDIDATES=len(unique\_candidates)

#print(NUM\_CANDIDATES)

# checking to see if can print each candidate correctly. change set to list

#unique\_candidates\_list = list(unique\_candidates)

# Sort alphabetically based on the names and print. use this later once calc votes for each above this line

#candidates\_sorted = sorted(unique\_candidates\_list)

#for candidate in candidates\_sorted:

#    print(candidate)

# Display the vote count for each candidate. calc percentage before this and add to the below to display too

for candidate, votes in candidate\_votes.items():

    percentage\_vote = votes/NUM\_VOTES\*100

    print(f"{candidate}: {percentage\_vote}% ({votes})")

#Calculate winner based on maximum votes. Use dictonary key ID to ID name of candidate bc using dictionary

winner = max(candidate\_votes, key=candidate\_votes.get)

print("Winner: ", winner)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)

**To make the decimals shorter in the percentage\_vote result, you can use the :.2f format specifier in the f-string to round the percentage to two decimal places.** Here's how you can modify the print statement for displaying the percentage of votes for each candidate:

for candidate, votes in candidate\_votes.items():

percentage\_vote = votes / NUM\_VOTES \* 100

print(f"{candidate}: {percentage\_vote:.2f}% ({votes})")

This change will ensure that the percentages are displayed with only two decimal places.

***Adjusted code using above so results of percentage votes is 3 decimal places out. It worked! Next work on exporting to .txt file***

#get file and use modules to read/use them

import os

import csv

pollpath = os.path.join('..','Resources','election\_data.csv')

#initialize starters. made an empty set where applicable

unique\_votes= set()

unique\_candidates= set()

candidate\_votes = {} #use dictironary to hold vote for each candidate

#add the rest of starters here, once do this step

# Reading using csv module. Open CSV file

with open(pollpath, encoding="utf-8") as pollfile:

    pollreader = csv.reader(pollfile)

    next(pollreader)  # Skip the header row

    for row in pollreader:

        ballot\_id\_column = row[0] #ballot id in 1st column

        vote = ballot\_id\_column.split(",")[0] #1st element after comma split

        unique\_votes.add(vote) #add unique vote to set after each loop

        candidate\_column=row[2] #3rd column

        candidate = candidate\_column.split(",")[0] #1st elmen after comma split?

        #note to self: I don't think split was necessary but was not sure how to add to set without this

        #later saw could do candidate = row[2]

        unique\_candidates.add(candidate) #add unique candidate to set after each loop

        #do loop to count each instance of each unique candidate

        if candidate in candidate\_votes:

            candidate\_votes[candidate] += 1

        else:

            candidate\_votes[candidate] = 1

        #loop to cal percentage of each

#Calculations based on above loops etc

#calc total votes

NUM\_VOTES=len(unique\_votes)

print("Total Votes: ",NUM\_VOTES)

#calc total candidates. do not need to print; just checking I did it correctly

#NUM\_CANDIDATES=len(unique\_candidates)

#print(NUM\_CANDIDATES)

# checking to see if can print each candidate correctly. change set to list

#unique\_candidates\_list = list(unique\_candidates)

# Sort alphabetically based on the names and print. use this later once calc votes for each above this line

#candidates\_sorted = sorted(unique\_candidates\_list)

#for candidate in candidates\_sorted:

#    print(candidate)

# Display the vote count for each candidate. calc percentage before this and add to the below to display too

for candidate, votes in candidate\_votes.items():

    percentage\_vote = votes/NUM\_VOTES\*100

    print(f"{candidate}: {percentage\_vote:.3f}% ({votes})")

#Calculate winner based on maximum votes. Use dictonary key ID to ID name of candidate bc using dictionary

winner = max(candidate\_votes, key=candidate\_votes.get)

print("Winner: ", winner)

# Specify the file to write to

#output\_path = os.path.join('..','Analysis','PyPoll\_Results.txt')

# Open the file using "write" mode. Specify the variable to hold the contents

#with open(output\_path, 'w') as file:

    #file.write("Election Results\n")

    #file.write("-------------------------\n")

    #file.write(f"Total Votes: {NUM\_VOTES}\n")

    #file.write("-------------------------\n")

    #fix so shows votes received by 1st candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 2nd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #fix so shows votes received by 3rd candidate file.write(f"Total: {TOTAL\_PROFIT\_LOSSES}\n")

    #file.write("-------------------------\n")

    #file.write(f"Winner: {[remove these brackets, insert variable that stores winner]}\n")

    #file.write("-------------------------\n")

#print("Results exported to PyPoll\_Results.txt")

# End-of-file (EOF)