**Task 2A**

**Solution 1 DEEPSEEK AI**

*"I’m working on a Python regex task for my DH course. I need to match multiple name variants from a list called names. I know I need to use re.escape() for literal matching, but I’m unsure how to combine them efficiently. Can you suggest an approach without giving me the full code?"*

**Response :**  
\*"Good question! Since you’re dealing with literal strings, re.escape() is indeed the right start. Think about how regex alternation works (hint: the | operator). How could you dynamically join all escaped names in your list into a single pattern? Try breaking it down:

1. How would you escape each name individually?
2. How would you combine them into one pattern?  
   Try writing a loop or a comprehension for this, and let me know where you get stuck."\*

**Code Comment for Your Script**

*# Student asked for guidance on combining regex patterns for name variants*

regex\_pattern = "|".join(re.escape(name) for name in names)

**explain how this code works**

Certainly! Let’s break down your code step by step to ensure you fully understand it and can justify its use in your assignment.

**Code Explanation:**

python

Copy

Download

regex\_pattern = "|".join(re.escape(name) for name in names)

**1. Purpose:**

This line constructs a **regex pattern** that will match *any* of the name variants stored in the names list. For example, if names = ["Paris", "London", "Berlin"], the resulting pattern will match "Paris" *or* "London" *or* "Berlin" in a text.

**2. Key Components:**

* re.escape(name):
  + **What it does:** Escapes special regex characters (e.g., \*, +, ?) in the name string so they’re treated as literal characters.
  + **Why it’s needed:** If a name contains regex metacharacters (e.g., "St. Petersburg" has .), re.escape() ensures they’re matched literally (not as regex wildcards).
  + **Example:** re.escape("St. Petersburg") returns "St\\. Petersburg".
* **Generator Expression**(re.escape(name) for name in names):
  + Iterates over each name in names, applies re.escape() to it, and yields the escaped strings one by one.
* "|".join(...):
  + Combines all escaped names into a single string, separated by the | (regex OR operator).
  + **Example Output:** If names = ["Paris", "London"], the result is "Paris|London".

**3. Final Regex Pattern:**

The resulting regex\_pattern can be used with re functions (e.g., re.findall()) to search for *any* of the names in a text.

**Hypothetical Example:**

python

Copy

Download

import re

names = ["New York", "St. Louis", "Los Angeles"]

regex\_pattern = "|".join(re.escape(name) for name in names)

*# Result: "New\ York|St\\.\ Louis|Los\ Angeles"*

text = "I visited New York and St. Louis last year."

matches = re.findall(regex\_pattern, text)

print(matches) *# Output: ['New York', 'St. Louis']*

Need to dissect another part of your code? I’m in ruthless tutor mode—ask away! 💥

**Solution 2**

i have written this code to count the number of times the name of a month was mentioned per month by looping through the month list which contains all the months where each name was mentioned #Print how many times the place was mentioned in each month by looping through it i want to print the output in an effective way suggest ways to do it

2/2

ChatGPT said:

Great! Since you've already looped through the month\_list and retrieved the count for each month, here are a few effective and clean ways you could print the output:

🟠 1. Key-Value Format (like a dictionary)

A tidy way to show the monthly counts:

python

CopyEdit

{ "January": 5, "February": 3, "March": 0, ... }

You could build this as part of your print block, something vaguely like:

python

CopyEdit

print(f' {month}: {count},') # inside the loop

🟠 2. Indented Multi-line Format

This gives each month a new line for clarity:

yaml

CopyEdit

Place A : {

January: 5

February: 3

March: 0

}

You might structure it loosely as:

python

CopyEdit

print(f" {month}: {count}") # again, within the loop

🟠 3. Tabular Format

Especially useful if comparing multiple places:

css

CopyEdit

Place January February March

Place A 5 3 0

This would need you to structure your print statements or use str.format() or f-strings with width formatting.

Would you prefer a more compact one-line format or something easier to read in blocks?

4o

You said:

`print(f' {month}: {count},') # inside the loop` \* \* \* ### 🟠 2. \*\*Indented Multi-line Format\*\* This gives each month a new line for clarity: yaml Copy code `Place A : { January: 5 February: 3 March: 0 }`

i prefer this one with this output. now explain how it works

ChatGPT said:

Perfect! You're referring to the Indented Multi-line Format, where each month's count is printed on a new line inside a dictionary-like block. Let's break down how it works step by step.

🔧 Code Structure (Vague Format)

Imagine you're printing something like:

yaml

CopyEdit

Place A : {

January: 5

February: 3

March: 0

}

Here's the general logic behind that:

1. Start the block for the place

You open the curly brace after printing the place name:

python

CopyEdit

print(f'{place} : {{') # ← This prints: Place A : {

2. Loop through each month

You loop through your month\_list, and for each month, get the count:

python

CopyEdit

for month in month\_list:

count = mentions\_per\_month[place][month] # retrieve the exact count

3. Print each month and its count indented

Within the loop, you print each month: count on a new line with indentation for readability:

python

CopyEdit

print(f' {month}: {count}') # 4 spaces of indentation

This results in lines like:

yaml

CopyEdit

January: 5

February: 3

4. Close the block

After the loop, you can optionally close the dictionary-style block:

python

CopyEdit

print('}')

🧠 Why It's Effective

* Human-readable: Clean layout, each line easy to scan.
* Scalable: Works well if you want to add more months or places.
* Semi-structured: Looks a bit like JSON or Python dicts, which makes it easy to debug or adapt for other formats.

Would you like a full example using fake data to test how this looks printed?

4o

You said:

the code is easy to follow but this is not what i am looking for. i want the output to print the place of the name and then the frequency of that name in each month through a nested dictinary can i adapt this code for this suggest many ways

ChatGPT said:

Got it! You're aiming for output that looks like a nested dictionary, something like:

python

CopyEdit

{

"Place A": {

"January": 5,

"February": 3,

"March": 0

},

"Place B": {

"January": 1,

"February": 0,

"March": 7

}

}

You can definitely adapt your loop to print in this format! Below are several ways to achieve this depending on how clean, compact, or programmatically useful you want it to be.

✅ WAY 1: Directly Print Nested Dictionary Format (Manual print)

python

CopyEdit

print('{')

for place in mentions\_per\_month:

print(f' "{place}": {{')

for month in month\_list:

count = mentions\_per\_month[place][month]

print(f' "{month}": {count},')

print(' },')

print('}')

🟡 Pros: Fully readable and customizable  
🔴 Cons: Manual formatting—need to manage commas, indenting

✅ WAY 2: Build a Nested Dictionary and Print It with print()

python

CopyEdit

nested\_dict = {}

for place in mentions\_per\_month:

nested\_dict[place] = {}

for month in month\_list:

nested\_dict[place][month] = mentions\_per\_month[place][month]

print(nested\_dict)

🟡 Pros: Real Python object you can reuse later, not just for display  
🔴 Cons: Less pretty formatting unless you use pprint

✅ WAY 3: Use json.dumps() to Format Nicely

python

CopyEdit

import json

nested\_dict = {}

for place in mentions\_per\_month:

nested\_dict[place] = {}

for month in month\_list:

nested\_dict[place][month] = mentions\_per\_month[place][month]

print(json.dumps(nested\_dict, indent=2))

🟢 Best option for a clean, real nested dictionary printout  
Looks like valid JSON and is very readable

✅ WAY 4: Use pprint for Pretty Printing

python

CopyEdit

from pprint import pprint

nested\_dict = {}

for place in mentions\_per\_month:

nested\_dict[place] = {}

for month in month\_list:

nested\_dict[place][month] = mentions\_per\_month[place][month]

pprint(nested\_dict)

🟡 Good for debugging in a Pythonic way  
🔴 Slightly less control over format than json.dumps()

Summary

| Goal | Recommended Option |
| --- | --- |
| Display-only, clean JSON-like | ✅ json.dumps() |
| Reusable dictionary + readable | ✅ pprint() or manual nested dict |
| Total control over spacing/commas | ✅ Manual print() formatting |

Would you like to test this with a mini example using fake data?

4o

You said:

print('{') for place in mentions\\_per\\_month: print(f' "{place}": {{') for month in month\\_list: count = mentions\\_per\\_month\[place\]\[month\] print(f' "{month}": {count},') print(' },') print('}')

explain how it works

ChatGPT said:

Great — let’s break down how this manual nested dictionary printing code works, line by line, so you can clearly understand how it builds the output.

You're essentially *mimicking the structure of a nested Python dictionary* using formatted print() statements.

🔁 Your Code Explained

python

CopyEdit

print('{')

* This prints the opening brace of the outer dictionary.
* You’re setting up a structure like:

python

CopyEdit

{

...

}

python

CopyEdit

for place in mentions\_per\_month:

print(f' "{place}": {{')

* This starts a loop through all the place names.
* For each place, you're printing its name as a key followed by a { to start its value, which will be another dictionary.
* The f' "{place}": {{' prints something like:

css

CopyEdit

"Place A": {

python

CopyEdit

for month in month\_list:

count = mentions\_per\_month[place][month]

print(f' "{month}": {count},')

* This inner loop goes through each month in month\_list.
* For the current place, you extract how many times it was mentioned in that month.
* Then you print that month and its count as a key-value pair inside the inner dictionary.
* Example output inside the inner loop:

arduino

CopyEdit

"January": 5,

"February": 3,

...

python

CopyEdit

print(' },')

* This closes the inner dictionary for one place, with a trailing comma to separate it from the next one.
* Output:

CopyEdit

},

python

CopyEdit

print('}')

* After all places and their months have been printed, this line closes the outer dictionary.

🧾 Final Output Example

When this runs on actual data, your console will print something like:

python

CopyEdit

{

"Place A": {

"January": 5,

"February": 3,

"March": 0,

},

"Place B": {

"January": 2,

"February": 0,

"March": 4,

},

}

This isn’t *technically* a real Python dictionary (because of the trailing commas), but it looks very close — and it's very readable.

Would you like a tip on how to avoid the trailing comma on the last item for a cleaner look?

4o

You said:

understood now i want to remove the comma after the last month

ChatGPT said:

Great! To remove the comma after the last month, you'll need to detect when you're printing the last item in the month\_list. Here are two clean ways to do it:

✅ OPTION 1: Use enumerate() + Index Check

This is the simplest and most readable method.

python

CopyEdit

for place in mentions\_per\_month:

print(f' "{place}": {{')

for i, month in enumerate(month\_list):

count = mentions\_per\_month[place][month]

# Check if it's the last month

if i == len(month\_list) - 1:

print(f' "{month}": {count}')

else:

print(f' "{month}": {count},')

print(' },')

✅ Easy to understand  
🔁 Just checks if you're at the last index and skips the comma

✅ OPTION 2: Use for month in month\_list[:-1] + Separate Last Item

Split the loop into two parts: one for all but the last, one for the last.

python

CopyEdit

for place in mentions\_per\_month:

print(f' "{place}": {{')

for month in month\_list[:-1]: # All except last

count = mentions\_per\_month[place][month]

print(f' "{month}": {count},')

# Now print the last month without a comma

last\_month = month\_list[-1]

count = mentions\_per\_month[place][last\_month]

print(f' "{last\_month}": {count}')

print(' },')

✅ Avoids conditional checks inside the loop  
🔄 Slightly longer, but very clean

Both options will give you output like this (no comma after "March"):

python

CopyEdit

{

"Place A": {

"January": 5,

"February": 3,

"March": 0

},

...

}

Would you like a version that also removes the comma after the last place in the outer dictionary?

4o

You said:

# Print the output by adding a comma after each line, except the last one for a proper format if month != month\_list[-1]: # adds comma if not the last month print(f' "{month}": {count},') else: print(f' "{month}": {count}') #End the dictionary block for this place and print the output print("},"). this is what i came up with

2/2

ChatGPT said:

Perfect — your code is exactly right for removing the comma after the last month! Let's go through it to reinforce why it works well.

✅ Your Code: Breakdown

python

CopyEdit

if month != month\_list[-1]: # adds comma if not the last month

print(f' "{month}": {count},')

else:

print(f' "{month}": {count}')

* You're comparing month to month\_list[-1], which is the last element in the list.
* If it's *not* the last, you add a comma; otherwise, you don't.
* This keeps your formatting clean and removes that annoying trailing comma after the final entry.

✅ Then you close the block:

python

CopyEdit

print("},") # Ends the dictionary for this place

* This closes the inner dictionary (per place), and the comma is fine here because multiple places might follow.

🧠 Small Tip (Optional)

If you also want to remove the comma after the last place, you can apply the same logic by checking:

python

CopyEdit

if place != list(mentions\_per\_month.keys())[-1]:

print(" },")

else:

print(" }")

Let me know if you'd like to integrate that as well!

**Task 4B**

**Solution 1**

i am making a map using two tsv files one containing the names of places in gaza with months and the frequency of their names per month in a corpus of articles and another tsv file which contains the coordintes for the places, i have to merge these two dataframes, but the problem is that in the first tsv file the column heading for the place names is placename and in the second file it is asciiname. but the names in both are the same. suggest me ways to merge the tables in a way that would contain a common column for the names, would it work if i rename the asciiname with placename

3/3

**ChatGPT said:**

Yes, you're absolutely right — renaming the asciiname column to placename is a clean and effective solution **if** the names in both files match exactly (same spelling, case, spacing, etc.). Here's a step-by-step approach to merging your two TSV files:

**✅ Step 1: Load the TSV Files**

python

CopyEdit

import pandas as pd

# Load frequency data (TSV with 'placename' column)

freq\_df = pd.read\_csv('frequency\_data.tsv', sep='\t')

# Load coordinates data (TSV with 'asciiname' column)

coords\_df = pd.read\_csv('coordinates\_data.tsv', sep='\t')

**✅ Step 2: Rename asciiname to placename**

python

CopyEdit

coords\_df.rename(columns={'asciiname': 'placename'}, inplace=True)

This ensures both dataframes have a common key for merging.

**✅ Step 3: Merge the DataFrames on placename**

python

CopyEdit

merged\_df = pd.merge(freq\_df, coords\_df, on='placename', how='inner')

* how='inner' keeps only rows that have a match in both dataframes.
* Use how='left' if you want to preserve all frequency data even if a coordinate is missing.

**✅ Optional: Normalize the placenames for safety**

If you suspect inconsistencies in spelling or casing, normalize before merging:

python

CopyEdit

freq\_df['placename'] = freq\_df['placename'].str.strip().str.lower()

coords\_df['placename'] = coords\_df['placename'].str.strip().str.lower()

Would you like help checking if there are any unmatched placenames between the two files?

4o

**You said:**

oords\\_df.rename(columns={'asciiname': 'placename'}, inplace=True)

what does the inplace = true does

**ChatGPT said:**

Great question.

The inplace=True argument in pandas means that the operation will be **applied directly to the original DataFrame** and **will not return a new DataFrame**.

So this line:

python

CopyEdit

coords\_df.rename(columns={'asciiname': 'placename'}, inplace=True)

does two things:

1. Renames the column 'asciiname' to 'placename' **inside** coords\_df.
2. Does **not** return anything (i.e., it returns None), because it modifies the DataFrame in-place.

If you leave out inplace=True, you'd need to **assign** the result to keep the changes:

python

CopyEdit

coords\_df = coords\_df.rename(columns={'asciiname': 'placename'})

So:

* inplace=True → modifies the original.
* inplace=False or omitted → returns a modified copy.

Would you prefer to keep the original dataframe untouched?

4o

**You said:**

If you leave out `inplace=True`, you'd need to \*\*assign\*\* the result to keep the changes:

what does this mean

**ChatGPT said:**

Let me break that down clearly:

In Python's pandas, most DataFrame methods return a **new copy** of the DataFrame with the changes applied — **unless** you tell it to apply the change directly using inplace=True.

So when I say:

If you leave out inplace=True, you'd need to **assign** the result to keep the changes:

It means this:

If you write this without inplace=True:

python

CopyEdit

coords\_df.rename(columns={'asciiname': 'placename'})

The rename **happens in a new temporary copy**, but it doesn’t change your original coords\_df. So if you try to use coords\_df afterward, the column will still be called 'asciiname'.

To actually save the renamed version, you'd need to do this:

python

CopyEdit

coords\_df = coords\_df.rename(columns={'asciiname': 'placename'})

This assigns the updated copy (with the renamed column) back into coords\_df.

In short:

* inplace=True → changes coords\_df directly.
* Without inplace=True → you must reassign to keep the changes.

Would you like me to show this with a quick example using sample data?

4o

**You said:**

if i use the inplace and change the origional data frame would it affect the output of other codes where the dataframe has been used before referring to the column as asciiname

**ChatGPT said:**

Yes — **excellent insight**, and you're absolutely right to be cautious here.

**🔥 Summary:**

If you use inplace=True to rename asciiname to placename in the original dataframe (coords\_df), then **any code that comes *after* that change and tries to use coords\_df['asciiname'] will break**, because that column no longer exists.

However, any code that comes *before* the renaming won't be affected — Python executes line by line, so only future references are impacted.

**🧠 Example:**

python

CopyEdit

print(coords\_df['asciiname']) # ✅ Works here

coords\_df.rename(columns={'asciiname': 'placename'}, inplace=True)

print(coords\_df['asciiname']) # ❌ This will raise a KeyError

print(coords\_df['placename']) # ✅ This will work now

**✅ Safe Practice:**

If you want to **keep the original column for other uses**, you have two options:

**Option 1: Create a copy before renaming**

python

CopyEdit

coords\_copy = coords\_df.copy()

coords\_copy.rename(columns={'asciiname': 'placename'}, inplace=True)

This way, you preserve coords\_df with asciiname for any earlier or alternative use.

**Option 2: Rename without inplace, assign to new variable**

python

CopyEdit

coords\_renamed = coords\_df.rename(columns={'asciiname': 'placename'})

Now you can use coords\_df as-is elsewhere, and coords\_renamed for merging.

**Task 4B**

**Solution 2**

the last step of my code is to print the output in this form regex\_map.html and regex\_map.png. suggest ways to do this

ChatGPT said:

To save your final output as both an interactive map (regex\_map.html) and a static image (regex\_map.png), you’ll likely be using a mapping library like Folium (for the HTML) and possibly selenium or folium-staticmap hacks (for the PNG), or using Plotly or Matplotlib + Basemap/Cartopy for both.

Here’s a step-by-step based on two common workflows:

✅ Option 1: Using Folium for HTML Map (Interactive)

python

CopyEdit

import folium

# Create the map centered at Gaza

m = folium.Map(location=[31.5, 34.5], zoom\_start=10)

# Plot points from merged\_df

for \_, row in merged\_df.iterrows():

folium.CircleMarker(

location=[row['latitude'], row['longitude']],

radius=row['frequency'], # or scale it

popup=row['placename'],

color='blue',

fill=True,

fill\_opacity=0.6

).add\_to(m)

# Save interactive map

m.save('regex\_map.html')

This gives you an interactive HTML map with zoom, hover, etc.

✅ Option 2: Save as PNG (Static Image)

Folium doesn’t natively support saving to PNG. You have two options here:

🔸 A. Use Selenium to take a screenshot

python

CopyEdit

from selenium import webdriver

import time

# Load the HTML in a headless browser

driver = webdriver.Chrome()

driver.get('file://' + os.getcwd() + '/regex\_map.html')

time.sleep(2) # wait for map to render

driver.save\_screenshot('regex\_map.png')

driver.quit()

Make sure:

* You have ChromeDriver installed.
* You run pip install selenium.

🔸 B. Use Plotly Instead (easier PNG export)

If you don’t need interactivity:

python

CopyEdit

import plotly.express as px

fig = px.scatter\_geo(

merged\_df,

lat='latitude',

lon='longitude',

text='placename',

size='frequency',

projection="natural earth",

title="Gaza Placename Frequency Map"

)

fig.write\_html("regex\_map.html")

fig.write\_image("regex\_map.png") # Needs kaleido

You’ll need:

bash

CopyEdit

pip install plotly kaleido

✅ TL;DR

* Use Folium + m.save() for .html.
* Use Plotly or Selenium to generate .png.

Would you like help deciding between Folium and Plotly based on your goals (interactive vs. static, styling, etc.)?