CSI3007 – Advanced Python Programming

Lab Activity 6 (01-08-2025)

Python Dictionary Quiz

NAME:VIBHU AADHAV,M

REGNO:22MID0141

LAB:6

GROUP MEMBERS:

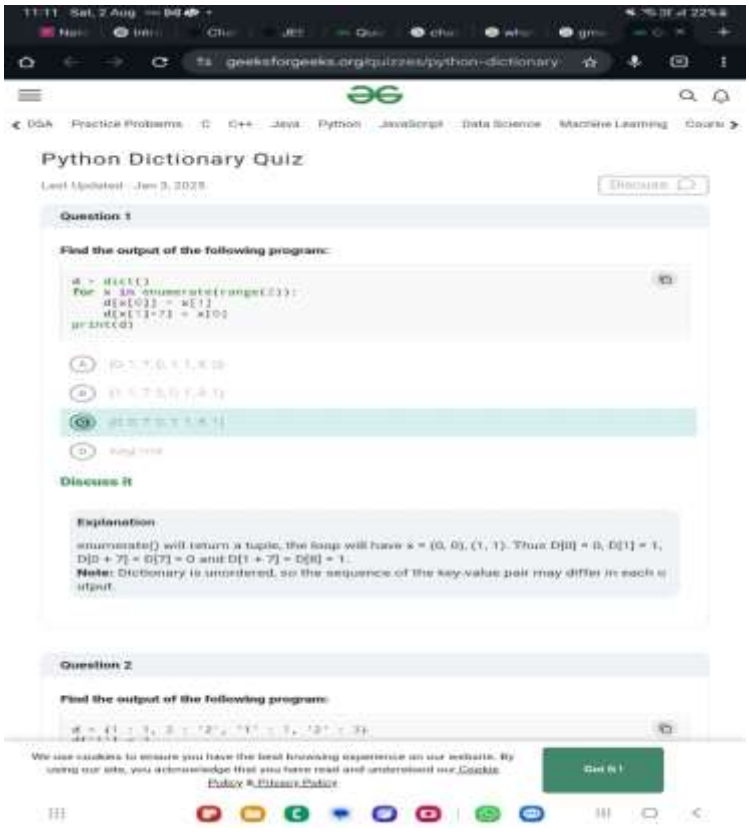
Varshitha V -22MID0105

Sidhdi Vijay K-22MIC0168

Gauvrav Sahoo - 22MIDO296

VIBHU AADHAV.M -22MID0141

Jewel -22MID0161




11:11 Sat, 2 Aug
DSA Practice Problems C C++ Java Python JavaScript Data Science Machine Learning Cours
Question 2
Find the output of the following program:
d = {1 : 1, 2 : '2', '1' : 1, '2' : 3}
d['1'] = 2
print(d[d[d[str(d[1])]]])
A 2
B 3
C 121
D KeyError
Discuss it
Explanation
Simple key-value pair is used recursively. D[1] = 1, str(1) = '1'. So, D[str(D[1])] = D['1'] = 2, D[2] = '2' and D['2'] = 3.
Question 3
Find the output of the following program:
d = {1 : {'A' : {1 : "A"}, 2 : "B"}, 3 :"C", 'B' : "D", "D": 'E'}
print(d[d[d[1][2]]], end = " ")
print(d[d[1]["A"][2]])
A C B
B E Key Error
C B D
D C E
We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy
Got It !

Question 4

Find the output of the following program:

```
d = dict()
for i in range (3):
    for j in range(2):
        d[i] = j
print(d)
```

(A) {0: 0, 1: 0, 2: 0}

(B) {0: 1, 1: 1, 2: 1}

(C) {0: 0, 1: 0, 2: 0, 0: 1, 1: 1, 2: 1}

(D) TypeError: Immutable object

Discuss it

Explanation

1st loop will give 3 values to i 0, 1 and 2. In the empty dictionary, valued are added and o verwrited in j loop, for eg. D[0] = [0] becomes D[0] = 1, due to overwriting.

Question 5

Question 5:Find the output of the following program:

```
d = {1 : [1, 2, 3], 2: (4, 6, 8)}
d[1].append(4)
print(d[1], end = " ")
li = [d[2]]
li.append(10)
d[2] = tuple(L)
print(d[2])
```

(A) [1, 2, 3, 4] ((4, 6, 8), 10)

(B) [1, 2, 3, 4] (4, 6, 8, 10)

(C) [1, 2, 3, 4] TypeError: tuples are immutable

Question 6

What will be the output of the following code?

```
s = "GeeksforGeeks"
print(s[0], s[-1])
```

A) G G

B) G s

C) G k

D) e k

Discuss it

Explanation

s[0] accesses the first character 'G' and s[-1] accesses the last character 's'.

Question 7

Find the output of the following program:

```
a = {}
a.fromkeys(['a', 'b', 'c', 'd'], 98)
print (a)
```

A) Syntax error

B) {'a':98, 'b':98, 'c':98, 'd':98}

C) {'a':None, 'b':None, 'c':None,'d':None}

D) {}

Discuss it

Question 8

Find the output of the following program:

```
dict ={}
print (all(dict))
```

(A) {}

(B) False

(C) True

(D) An exception is thrown

Discuss it

Explanation

The all() method returns:

True – If all elements in an iterable are true ot iterable is empty.
False – If any element in an iterable is false.

Question 9

Find the output of the following program:

```
a = {'geeks' : 1, 'gfg' : 2}
b = {'geeks' : 2, 'gfg' : 1}
print (a == b)
```

(A) True

(B) False

(C) Error

(D) None

Question 9

Find the output of the following program:

```
a = {'geeks' : 1, 'gfg' : 2}
b = {'geeks' : 2, 'gfg' : 1}
print (a == b)
```

(A) True

(B) False

(C) Error

(D) None

Discuss it

Explanation

If two dictionary are the same it returns true, otherwise it returns false.

Question 10

Which of the following is FALSE about dictionary?

(A) The values of a dictionary can be accessed using keys

(B) The keys of a dictionary can be accessed using values

(C) Both of the above

(D) None of the above

Discuss it

LAB ACTIVITY 7

NAME:VIBHU AADHAV.M

REGNO:22MID0141

1. Passing Arguments - Positonal Arguments and keyword Arguments

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet"""
    print(f"\n I have a pet {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet("Hamster","Harry")  # function call
```

```
 I have a pet Hamster.
My Hamster's name is Harry.
```

2.Multiple Function Call

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet"""
    print(f"\n I have a pet {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet("Hamster","Harry")  # function call
describe_pet("Dog","Bruno")      # Another function call
```

```
 I have a pet Hamster.
My Hamster's name is Harry.

 I have a pet Dog.
My Dog's name is Bruno.
```

3. Order matters in positional Arguments

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet"""
    print(f"\n I have a pet {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet("Harry","Hamster")  # function call
```

```
 I have a pet Harry.
My Harry's name is Hamster.
```

4. Keyword Arguments

```
def describe_pet(pet_name, animal_type = "dog"):
    """Display information about a pet"""
    print(f"\n I have a pet {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(pet_name = "Bruno") # function call
```

```
 I have a pet dog.
My dog's name is Bruno.
```

6. Hence many ways to call a function

```
describe_pet("Bruno") # Positional argument and default values
describe_pet(pet_name = "Bruno") # Keyword argument and default values
describe_pet(animal_type = "Hamster",pet_name = "Harry") # Keyword argument (doesn't matter on order)
describe_pet("Harry","Hamster") # Positional Argument
describe_pet("Hamster",animal_type="Harry") # Positional argument and Keyword argument
describe_pet(pet_name = "Harry",animal_type = "Hamster") # Keyword argument
```

```
 I have a pet dog.
My dog's name is Bruno.

 I have a pet dog.
My dog's name is Bruno.

 I have a pet Hamster.
My Hamster's name is Harry.

 I have a pet Hamster.
My Hamster's name is Harry.

 I have a pet Harry.
My Harry's name is Hamster.

 I have a pet Hamster.
My Hamster's name is Harry.
```

7. Returning a simple value

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted"""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi','hendrix')
print(musician)
```

```
Jimi Hendrix
```

8. Making an argument optional

8. Making an argument optional

```
# Function to return a full name, where the middle name is optional
def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted. Middle name is optional."""
    if middle_name:
        # If middle name is provided, include it in the full name
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        # If no middle name, just use first and last name
        full_name = f"{first_name} {last_name}"

    return full_name.title()  # Return the full name in title case (e.g., Jimi Hendrix)

# Calling the function with only first and last name
musician = get_formatted_name('jimi', 'hendrix')
print(musician)

# Calling the function with a middle name
musician = get_formatted_name('john', 'doe', 'paul')
print(musician)
```

```
Jimi Hendrix
John Paul Doe
```

9. Returning a dictionary

```
def get_formatted_name(first_name, last_name):
    """Return a dictionary with full name details"""
    return {
        'first_name': first_name.title(),
        'last_name': last_name.title()
    }

musician = get_formatted_name('jimi','hendrix')
print(musician)
```

```
{'first_name': 'Jimi', 'last_name': 'Hendrix'}
```

10. Passing a list

```
def greet_user(names):
    # printing a list of elements
    for name in names:
        msg = f"Hello, {name}"
        print(msg)

names = ['Hari', 'Sathish', 'Aru', 'Vasanth']
greet_user(names)
```

```
Hello, Hari
Hello, Sathish
Hello, Aru
Hello, Vasanth
```

11. Modifying a list

```
def greet_and_store(names, greeted_names):
    # Greet each user and move them to greeted_names list
    while names:
        name = names.pop(0)  # Remove from original list (from front)
        print(f"Hello, {name}")
        greeted_names.append(name)  # Add to greeted list

# Original list of names
names = ['Hari', 'Sathish', 'Aru', 'Vasanth']
greeted_names = []

# Call the function
greet_and_store(names, greeted_names)

# Display greeted users
print("\nGreeted users:")
print(greeted_names)
```

```
Hello, Hari
Hello, Sathish
Hello, Aru
Hello, Vasanth

Greeted users:
```

```
greeted_names = []

# Pass a copy of the list using names[:]
greet_and_store(names[:], greeted_names)

# Show the original and modified lists
print("\nOriginal list:")
print(names)  # Unchanged

print("\nGreeted users:")
print(greeted_names)  # Only contains greeted names
```

```
Hello, Hari
Hello, Sathish
Hello, Aru
Hello, Vasanth

Original list:
['Hari', 'Sathish', 'Aru', 'Vasanth']

Greeted users:
['Hari', 'Sathish', 'Aru', 'Vasanth']
```

13. Passing an Arbitrary number of arguments

```
def make_pizza(*toppings):
    # printing n number of arguments
    print(toppings)

make_pizza("pepperoni")
make_pizza("pepperoni","mushrooms", "green pepper","extra cheese")
```

```
('pepperoni',)
('pepperoni', 'mushrooms', 'green pepper', 'extra cheese')
```

14. Mixing Positional and Arbitrary Arguments

```
def make_pizza(size, *toppings):
    # size is a postional and toppings is a keyword argument
    print(f"\n Making a {size}-inch pizza with the following toppings")
    for topping in toppings:
        print(f"- {topping}")

make_pizza(16,"pepperoni")
make_pizza(22,"pepperoni","mushrooms", "green pepper","extra cheese")
```

```
 Making a 16-inch pizza with the following toppings
- pepperoni

 Making a 22-inch pizza with the following toppings
- pepperoni
- mushrooms
- green pepper
- extra cheese
```

LAB-8:

LAB-8:(05/08/2025)

Lambda Function

```
# create function using def
 keyword def double(x):
 return x*2
double(10) # function call
```

```
# create function using def keyword
def add(x,y):
    return x+y
add(25, 25)  # function call
```

```
50
```

```
# create fuction using lambda
x = lambda x,y:x+y
x(18,12) # function call
```

```
30
```

```
# create function using def keyword
def max(x,y):
    if x>y:
        return x
    else:
        return y
max(10,15) # function call
```

```
15
```

```
# create fuction using lambda
x = lambda x,y:x if x>y else y
x(7, 18) # function call
```

```
18
```

Iterables

```
num = [1,2,3,4,5]  # List is iterable
for i in num:
    print(i)
```

```
1
2
3
4
5
```

```
num1 = (1,2,3,4,5)  # tuple is iterable
for i in num1:
    print(i)
```

```
1
2
3
4
5
```

LAB-9 (08/08/2025)

```
# create a class (that do nothing)
class Emp:
    pass
```

```
e1 = Emp() # create a object of a class
```

```
print(e1)  # printing the object of the class
```

```
<__main__.Emp object at 0x00000157A26A7790>
```

```
print(Emp)  # printing the Class type
```

```
<class '__main__.Emp'>
```

```
# Assingning attributes to the class
e1.first_name = "Sathish"
e1.last_name = "Veera"
```

```
# Printing the attributes of the class
print(e1.first_name)
print(e1.last_name)
```

```
Sathish
Veera
```

```
# create a class (that contains attributes and methods)
class Employee:
    def __init__(self, first_name, last_name):          # first_name, last_name and email are calles as instance variables (attributes)
        self.first_name = first_name
        self.last_name = last_name
        self.email = f"{self.first_name}1@gmail.com"
    def fullname(self):
        return f"{self.first_name} {self.last_name}"     # printing the full name of the employee
```

```
        self.email = f"{first_name}.{last_name}@company.com"

        # Increment the employee count every time a new employee is created
        Employee.num_of_employees += 1

    def fullname(self):
        return f"{self.first_name} {self.last_name}"
```

```
# object creation and accessing the members and member functions of the class
print(f"Initial employee count: {Employee.num_of_employees}")  # Accessing the class variable

emp_1 = Employee('Elon', 'Musk', 50000)
emp_2 = Employee('Donald', 'Trump', 60000)
```

```
Initial employee count: 0
```

```
# Accessing the class variable
print(f"Current employee count: {Employee.num_of_employees}")
```

```
Current employee count: 2
```

```
# printing the name of the employees
print(emp_1.fullname())
print(emp_2.fullname())
```

```
Elon Musk
Donald Trump
```

LAB-10: (26-08-2025)

Forgetting Cursive Writing – A Google Ngram Based Analysis

Task : Fetch Google Books Ngram JSON, save as CSV, plot, generate word cloud and prepare a prompt for an LLM to produce a human summary.

```
In [10]: # Install necessary packages
         !pip install requests

Requirement already satisfied: requests in c:\users\sonal\appdata\roaming\python\python312\site-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from
requests) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from requests)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from reques
ts) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from reques
ts) (2025.1.31)
WARNING: Ignoring invalid distribution ~umpy (C:\Users\sonal\AppData\Roaming\Python\Python312\site-packages)
WARNING: Ignoring invalid distribution ~umpy (C:\Users\sonal\AppData\Roaming\Python\Python312\site-packages)

[notice] A new release of pip is available: 24.3.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

In [14]: !pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.9.4-cp312-cp312-win_amd64.whl.metadata (3.5 kB)
Requirement already satisfied: numpy>=1.6.1 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from wordcloud)
(1.26.4)
Requirement already satisfied: pillow in c:\users\sonal\appdata\roaming\python\python312\site-packages (from wordcloud) (11.1.
0)
Requirement already satisfied: matplotlib in c:\users\sonal\appdata\roaming\python\python312\site-packages (from wordcloud) (3.
9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from matplotl
ib->wordcloud) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from matplotlib->
wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from matplot
lib->wordcloud) (4.55.8)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from matplot
lib->wordcloud) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\sonal\appdata\roaming\python\python312\site-packages (from matplotli
b->wordcloud) (24.2)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

```
import json
import sys
```

In [23]:
```
# List of search terms (trends to analyze in Google Books Ngram Viewer)
TERMS = ["cursive", "penmanship", "handwriting"]   # Example: missing trends in millennials/Gen Z lifestyles

# Start year of analysis (earliest year to include in results)
YEAR_START = 1800

# End year of analysis (latest year to include in results)
YEAR_END = 2025

# Google Books Ngram corpus ID (15 = English, modern standard corpus)
# Other IDs correspond to different languages/corpora (check Ngram Viewer UI for details)
CORPUS = 15

# Smoothing factor (0 = no smoothing, higher numbers smooth out yearly fluctuations)
SMOOTHING = 0

# HTTP User-Agent header (used to mimic a browser request when fetching data)
USER_AGENT = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100 Safari/537.36"

# Filepath for saving extracted Ngram data in CSV format
OUTPUT_CSV = "ngrams_output.csv"

# Filepath for saving the generated trend plot as a PNG image
PLOT_PNG = "ngrams_plot.png"

# Filepath for saving the generated word cloud of terms as a PNG image
WORDCLOUD_PNG = "ngrams_wordcloud.png"
```

In [27]:
```
def fetch_ngram_json(terms, year_start=YEAR_START, year_end=YEAR_END, corpus=CORPUS, smoothing=SMOOTHING):
    """
    Uses the Google Ngram JSON endpoint:
    https://books.google.com/ngrams/json?content=...&year_start=...&year_end=...&corpus=...&smoothing=...
    Returns parsed JSON (list of dicts) on success, raises on failure.
    """
    url = "https://books.google.com/ngrams/json?content=cursive,penmanship,handwriting&year_start=1800&year_end=2025&corpus=15
    params = {
        "content": ",".join(terms),
```

```
        "year_start": year_start,
        "year_end": year_end,
        "corpus": corpus,
        "smoothing": smoothing,
    }
    headers = {"User-Agent": USER_AGENT}
    r = requests.get(url, params=params, headers=headers, timeout=30)
    r.raise_for_status()
    # The endpoint returns JSON that pandas.read_json can also parse; here we return Python list/dict
    return r.json()
```

In [29]:
```
def json_to_dataframe(ngram_json, year_start=YEAR_START, year_end=YEAR_END):
    """
    Convert the JSON returned by the endpoint into a tidy pandas DataFrame:
    columns = ['year', 'ngram', 'freq']
    """
    years = list(range(year_start, year_end + 1))
    rows = []
    for series in ngram_json:
        ngram = series.get("ngram")
        timeseries = series.get("timeseries", [])
        if len(timeseries) != len(years):
            # fill/truncate defensively
            timeseries = (timeseries + [0] * len(years))[:len(years)]
        for y, v in zip(years, timeseries):
            rows.append({"year": y, "ngram": ngram, "freq": float(v)})
    df = pd.DataFrame(rows)
    return df
```

```
    print("[!] Error fetching Ngram JSON:", e)

    # Exit the script safely with error code 1
    sys.exit(1)
```

```
[*] Fetching Ngram JSON for: ['cursive', 'penmanship', 'handwriting']
[+] Successfully fetched Ngram data for 3 terms.
```

In [51]:
```
# Convert the fetched Ngram JSON into a long-format DataFrame
# (columns: year, term, frequency) for easier analysis/plotting
df_long = json_to_dataframe(ngram_json)
```

In [53]:
```
# Pivot the long DataFrame into wide format (rows = years, columns = terms, values = frequencies)
pivot_df = pivot_timeseries(df_long)
```

In [55]:
```
# Save the pivoted DataFrame to a CSV file for later use
save_csv(pivot_df)
```

```
[+] Saved CSV to ngrams_output.csv
```

In [57]:
```
# Generate and display a time series plot of term frequencies from the pivoted DataFrame
plot_timeseries(pivot_df)
```

```
[+] Saved time-series plot to ngrams_plot.png
```

In [59]:
```
# Generate and save a word cloud image from the pivoted DataFrame
generate_wordcloud(pivot_df)
```

```
[+] Saved word cloud to ngrams_wordcloud.png
```

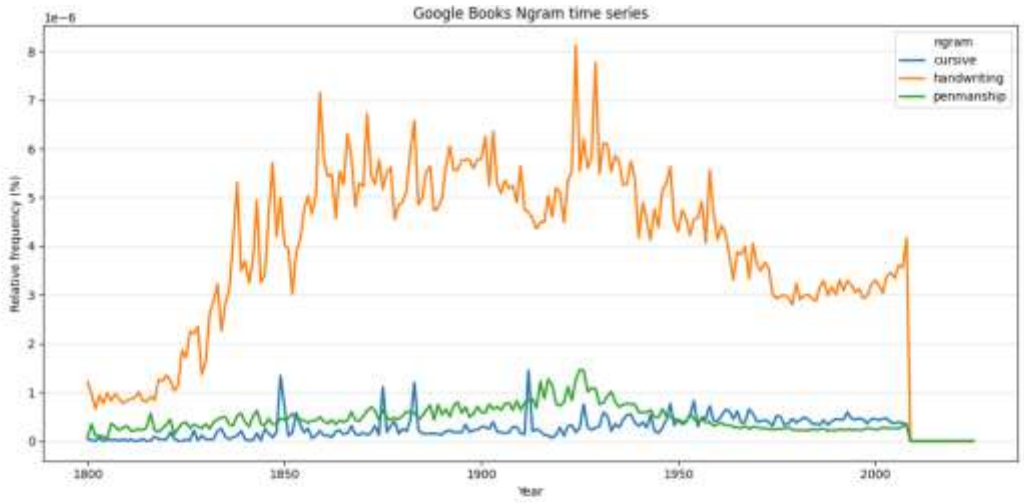
In [61]:
```
# Compute summary statistics (e.g., mean, max, trends) for each term
stats = compute_summary_stats(pivot_df)
```

In [63]:
```
# Print per-term summary statistics in a readable JSON format
print("\n[Summary statistics (per-term)]:")
print(json.dumps(stats, indent=2))
```

In [69]:
```
# Final confirmation with names of all saved output files
print("\n[Done] Files: ", OUTPUT_CSV, PLOT_PNG, WORDCLOUD_PNG)
```

```
[Done] Files:  ngrams_output.csv ngrams_plot.png ngrams_wordcloud.png
```

In [135…
```
# Visualization of time series plot of term frequencies from the pivoted DataFrame
plt.figure(figsize=(12,6))
ax = plt.gca()
pivot_df.plot(ax=ax, linewidth=2)
ax.set_xlabel("Year")
ax.set_ylabel("Relative frequency (%)")
ax.set_title("Google Books Ngram time series")
ax.grid(axis="y", alpha=0.3)
plt.legend(title="ngram", loc="upper right")
plt.tight_layout()
```



In [139...

```
# word cloud from the pivoted DataFrame
means = pivot_df.mean(axis=0).to_dict()
# WordCloud expects a text or a frequency dict; we feed frequency dict
wc = WordCloud(width=800, height=400, background_color="white")
wc.generate_from_frequencies(means)
plt.figure(figsize=(10,5))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud (from LLM-generated keywords)")
plt.show()
```