Code Explanation: Generating a Math Assessment Word Document

This document explains a Python script designed to run in Google Colab, which generates a Microsoft Word document (`Generated_Math_Questions.docx`) containing two math questions aligned with the Quantitative Math curriculum. The questions include a combinatorics problem with a table and a geometry problem with an image of tightly packed spheres. The script uses the Google Gemini API to generate questions, `matplotlib` to create the image, and `python-docx` to build the document, which is then downloaded and can be shared via Google Docs and GitHub.

Step 1: Install Required Libraries

```
!pip install python-docx matplotlib google-generativeai
```

- Installs three Python libraries using a shell command in Google Colab:
- `python-docx`: For creating and editing Word (.docx) documents.
- `matplotlib`: For generating the packed spheres image.
- `google-generativeai`: For accessing the Gemini API to generate math questions.

Step 2: Import Libraries

```
import os
import re
import json
import google.generativeai as genai
from docx import Document
from docx.shared import Inches
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from google.colab import files
from google.colab import userdata
```

- Imports modules for:
- `os`: File system operations (minimal use).
- `re`: Regular expression parsing for the image description.
- `json`: Parsing the Gemini API's JSON response.
- `google.generativeai`: Interacting with the Gemini API.
- `docx.Document` and `docx.shared.Inches`: Creating and formatting Word documents.

- `matplotlib.pyplot` and `matplotlib.patches.Circle`: Generating the sphere image.
- `google.colab.files`: Downloading the document in Colab.
- `google.colab.userdata`: Accessing the API key from Colab Secrets.

Step 3: Configure Gemini API

```
def configure_gemini_api():
    try:
        api_key = userdata.get('GOOGLE_GEMINI_API_KEY')
        if not api_key:
            raise ValueError("GOOGLE_GEMINI_API_KEY not found in
               Colab Secrets.")
        genai.configure(api_key=api_key)
    except Exception as e:
        print(f"Error configuring Gemini API: {e}")
        print("Please set the API key in Colab Secrets with name '
           GOOGLE_GEMINI_API_KEY'.")
        raise
```

- Defines a function to configure the Gemini API.
- Retrieves the API key from Colab Secrets using `userdata.get`.
- Raises an error if the key is missing, with instructions to set it in Colab's Secrets tab.
- Configures the API with `genai.configure`.
- Includes error handling for invalid keys or network issues.

Step 4: Call Gemini API to Generate Questions

```
def call_gemini_api(prompt, image_params=None):
    try:
        model = genai.GenerativeModel("gemini-1.5-pro")
        if image_params:
            prompt += f"""
    Generate a geometry question about tightly packed spheres with
       the following characteristics:
    - Total number of spheres: {image_params['num_spheres']}
    - Arrangement grid: {image_params['grid_rows']} by {image_params
       ['grid_cols']}
    - Radius of each sphere: {image_params['radius']} centimeters
    """
        response = model.generate_content(prompt)
        response_text = response.text
        if response_text.startswith("```json") and response_text.
           endswith("```"):
```

```
            response_text = response_text[7:-3].strip()
        return json.loads(response_text)
    except Exception as e:
        print(f"Error calling Gemini API: {e}")
        return { ... }  % Hardcoded fallback response
```

- Defines a function to call the Gemini API and generate two math questions in JSON format.
- Uses the `gemini-1.5-pro` model.
- Appends image parameters (e.g., 9 spheres, 3x3 grid, radius 2 cm) to the prompt if provided.
- Processes the response, removing markdown code block markers if present.
- Parses the response as JSON to extract questions with fields like `question`, `table`, `image`, etc.
- Returns a hardcoded fallback response if the API call fails.

Step 5: Generate Image for the Geometry Question

```
def generate_packed_spheres_image(image_description):
    match = re.match(r"Generate a top view of (\d+) spheres arranged
        in a (\d+) by (\d+) grid, each with radius (\d+) cm\.",
       image_description)
    if not match:
        print("Warning: Could not parse image description. Using
           default 2x3 grid with radius 2 cm.")
        rows, cols, radius = 2, 3, 2
    else:
        _, rows, cols, radius = map(int, match.groups())
    fig, ax = plt.subplots(figsize=(5, 3))
    ax.set_aspect('equal')
    center_spacing = 2 * radius
    for row in range(rows):
        for col in range(cols):
            center_x = col * center_spacing
            center_y = row * center_spacing
            circle = Circle((center_x, center_y), radius, edgecolor=
               'black', facecolor='none', linewidth=1)
            ax.add_patch(circle)
    ax.set_xlim(-radius, cols * center_spacing + radius)
    ax.set_ylim(-radius, rows * center_spacing + radius)
    ax.axis('off')
    plt.savefig('packed_spheres.png', bbox_inches='tight', dpi=200)
    plt.close()
```

- Generates an image of tightly packed spheres based on the `image` field (e.g., "Generate a top view of 9 spheres arranged in a 3 by 3 grid, each with radius

2 cm.”).

- Uses regex to parse the number of spheres, grid dimensions, and radius.
- Defaults to a 2x3 grid with radius 2 cm if parsing fails.
- Creates a `matplotlib` figure (5x3 inches) with circles spaced by the diameter.
- Saves the image as `packed_spheres.png` with 200 DPI.

Step 6: Create the Word Document

```
def create_word_document(questions):
    doc = Document()
    doc.add_paragraph('@title Math Assessment: Combinatorics and
       Geometry Problems')
    doc.add_paragraph('@description This assessment contains two
       generated math questions ...')
    for q in questions:
        doc.add_paragraph(f'// Question {q["order"]}')
        doc.add_paragraph(f'@question {q["question"]}')
        if "table" in q:
            doc.add_paragraph('## Meal Choices')
            table = doc.add_table(rows=len(q["table"]["rows"]) + 1,
               cols=2)
            table.style = 'Table Grid'
            ...
        if "image" in q:
            generate_packed_spheres_image(q["image"])
            doc.add_picture('packed_spheres.png', width=Inches(3))
        ...
    doc.save('Generated_Math_Questions.docx')
    files.download('Generated_Math_Questions.docx')
    print("The document is being downloaded. To share, upload it to
       Google Docs and GitHub ...")
```

- Creates a Word document with `python-docx`.
- Adds a title and description.
- For each question:
- Adds the question text, table (if present), or image (if specified).
- Embeds `packed_spheres.png` for the geometry question.
- Adds options (correct option marked with `@@option`) and metadata.
- Saves and downloads the document.

Step 7: Main Execution

```
if __name__ == "__main__":
    configure_gemini_api()
    prompt = """
    Generate two math questions in JSON format, similar to the
       following base questions ...
    """
    new_image_params = {
        'num_spheres': 9,
        'grid_rows': 3,
        'grid_cols': 3,
        'radius': 2
    }
    llm_response = call_gemini_api(prompt, image_params=
       new_image_params)
    create_word_document(llm_response["questions"])
```

- Runs the script, configuring the API, generating questions, and creating the document.
- Specifies a 3x3 grid with radius 2 cm for the geometry question.

Expected Output

- **Word Document**: `Generated_Math_Questions.docx` with two questions, a table, an image, and metadata.
- **Image**: A 3x3 grid of circles (radius 2 units) saved as `packed_spheres.png`.
- **Sharing**: Upload the document to Google Docs and GitHub, then share the links.

Notes

- Set `GOOGLE_GEMINI_API_KEY` in Colab Secrets.
- Ensure the Gemini API's image description matches the 3x3 grid, radius 2 cm.
- LaTeX equations may require a compatible viewer in Word.