ASSIGNMENT_08

Practice and understand all the code snippets provided below:

```
!pip install google-generativeai PyPDF2 gtts pyaudio
```

```
# Install system dependencies first (Colab-specific)
!apt-get update
!apt-get install -y portaudio19-dev python3-pyaudio

# Then install pyaudio
!pip install pyaudio
```

```
# Import libraries
import google.generativeai as genai
import getpass

# Input your Gemini API key securely
api_key = getpass.getpass("🔑 Enter your GEMINI_API_KEY: ")

# Configure Gemini API
if not api_key:
    print("⚠️  No API key provided.")
    print("💡 Get your API key from: https://makersuite.google.com/app/apikey")
else:
    try:
        genai.configure(api_key=api_key)
        print("✅ Gemini API configured successfully!")

        # Test the connection
        model = genai.GenerativeModel('gemini-2.5-flash')
        response = model.generate_content("Say 'Hello from Gemini!' in a fun way, in 1 line and simple text, no md")
        print(f"🤖 Gemini says: {response.text}")

    except Exception as e:
        print(f"❌ Error connecting to Gemini: {e}")
        print("💡 Make sure your API key is correct and you have internet connection")
```

```python
# 🎯 STEP 1: Basic Chat Completion (The Foundation)
print("🚀 STEP 1: Basic Chat Completion")
print("=" * 50)

# Simple chat function
def chat_with_ai(message):
    """Basic chat completion with Gemini"""
    try:
        response = model.generate_content(message)
        return response.text
    except Exception as e:
        return f"Error: {e}"

# Test basic chat
print("💬 Testing basic chat:")
response = chat_with_ai("I want to hear about AI in 20 words, I want an audio file in output")
print(f"AI: {response}")

print("\n✅ Basic chat working! This is the foundation for all our amazing apps!")
```

```python
# 🎯 STEP 2: Streaming Chat
import time
import sys

print("🚀 STEP 2: Streaming Chat - Watch AI think in real-time!")
print("=" * 60)


def streaming_chat(prompt, token_delay=0.05):
    """
    Stream Gemini output chunk-by-chunk, and simulate token-by-token display.
    """
    print("💬 Prompt sent. AI is responding...\n")
    full_response = ""
-
    try:
        response = model.generate_content(prompt, stream=True)

        for chunk in response:
            text = chunk.text or ""
            if text:
                for token in text.split():
                    print(token + " ", end="", flush=True)
                    sys.stdout.flush()
                    time.sleep(token_delay)
                full_response += text

        print("\n\n✅ Done streaming.")
        return full_response

    except Exception as e:
        print(f"❌ Error during streaming: {e}")
        return ""

# Test it
streaming_chat("I want to hear about AI in 20 words")
```

```python
# 🎯 STEP 3: PDF Chat Assistant (Talk to any uploaded PDF!)
print("🚀 STEP 3: PDF Chat Assistant - Talk to any uploaded PDF document!")
print("=" * 70)

import PyPDF2
from google.colab import files
import io

# Upload PDF file
uploaded = files.upload()

# Get the uploaded file path
pdf_filename = next(iter(uploaded))  # Get the first uploaded file

def read_pdf_text(file_obj):
    """Extract text from uploaded PDF file object"""
    try:
        pdf_reader = PyPDF2.PdfReader(file_obj)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {e}"

def chat_with_pdf(pdf_text, question):
    """Ask questions about PDF content using Gemini"""
    try:
        prompt = f"""
        Based on the following PDF content, answer this question: {question}

        PDF Content:
        {pdf_text[:3000]}  # Only first 3000 characters to keep prompt short

        Please provide a clear and accurate answer based on the PDF content.
        """
        response = model.generate_content(prompt)
        return response.text
    except Exception as e:
        return f"Error: {e}"

# Read the PDF text
with open(pdf_filename, 'rb') as file_obj:
    pdf_text = read_pdf_text(file_obj)

# Ask a question
question = "How many transformeres have been asked for the quotation!"
answer = chat_with_pdf(pdf_text, question)

# Display results
print(f"\n❓ Question: {question}")
print(f"🤖 AI Answer: {answer}")

print("\n✅ PDF Chat is working! You can ask any question about the uploaded PDF.")
```

```python
from gtts import gTTS

def chat(message):
    return model.generate_content(message).text

def tts(text):
    gTTS(text).save("ai_speech.mp3")
    print(f"🎵 Audio saved: {text[:50]}...")

tts(chat("Tell me about space"))
```

```python
import json
from gtts import gTTS
from IPython.display import Audio
import google.generativeai as genai

# Tool implementations
def tts_tool(text):
    gTTS(text).save("output.mp3")
    print("🔊 Playing audio...")
    return Audio("output.mp3")

def text_output_tool(text):
    print("📄 Text Output:")
    print(text)
    return text

# Tool definitions for prompt context
tool_descriptions = [
    {
        "name": "tts_tool",
        "description": "Convert text to speech when user asks to hear something, such as pronunciation or read-aloud.",
        "parameters": {
            "type": "object",
            "properties": {
                "text": {"type": "string"}
            },
            "required": ["text"]
        }
    },
    {
        "name": "text_output_tool",
        "description": "Display information as plain text when no audio is requested.",
        "parameters": {
            "type": "object",
            "properties": {
                "text": {"type": "string"}
            },
            "required": ["text"]
        }
    }
]
```

```python
# 🗣️ Ask user
user_input = input("You: ")

# System prompt with strict formatting instruction
system_prompt = f"""
You are a smart assistant with two tools available:

1. tts_tool: Use when the user wants to HEAR the response (e.g. pronunciation, "say", "read aloud", etc).
2. text_output_tool: Use when the user wants written info (e.g. facts, definitions, etc).

Respond ONLY with a valid JSON object like:
{{"tool_name": "...",
  "parameters": {{"text": "..."}}
}}

DO NOT include explanations, markdown, or anything else.

User: "{user_input}"
"""

# Generate content from Gemini
response = model.generate_content(system_prompt)
response_text = response.text.strip()

# DEBUG: Print raw response if needed
print("Gemini raw response:", response_text)

# Safe JSON parsing
try:
    tool_call = json.loads(response_text)
    tool_name = tool_call["tool_name"]
    params = tool_call["parameters"]
except json.JSONDecodeError as e:
    print("❌ Gemini did not return valid JSON:")
    print(response_text)
    raise e

# 🔧 Run the selected tool
if tool_name == "tts_tool":
    output = tts_tool(params["text"])
    display(output)
elif tool_name == "text_output_tool":
    output = text_output_tool(params["text"])
else:
    print(f"❌ Unknown tool: {tool_name}")
```

Multi-Tool Application Challenge: Intelligent Travel Guide Generator

Objective:

Build a Python application that chains multiple AI tools to create a personalized travel guide with real-time weather integration.

Required Tool Chain:

1. User Input: Destination city and travel dates

2. Weather API: Fetch current weather and 5-day forecast (OpenWeatherMap free tier)
3. LLM Content Generation: Generate travel guide sections using weather data as context
4. Image Generation: Create 5+ relevant images for the guide
5. PDF Creation: Compile text and images into professional 5-7 page PDF
6. Audio Generation: Convert entire guide to audio narration

The application should demonstrate how output from one tool becomes input for the next, creating a seamless multi-tool workflow.