Project Overview

This document provides a detailed overview of the Python scripts in this project. Each section below describes a script's purpose, its pseudocode, and a Mermaid diagram illustrating its logic.

Core Content Generation

a01_RAG_DB_Creation_PDF.py

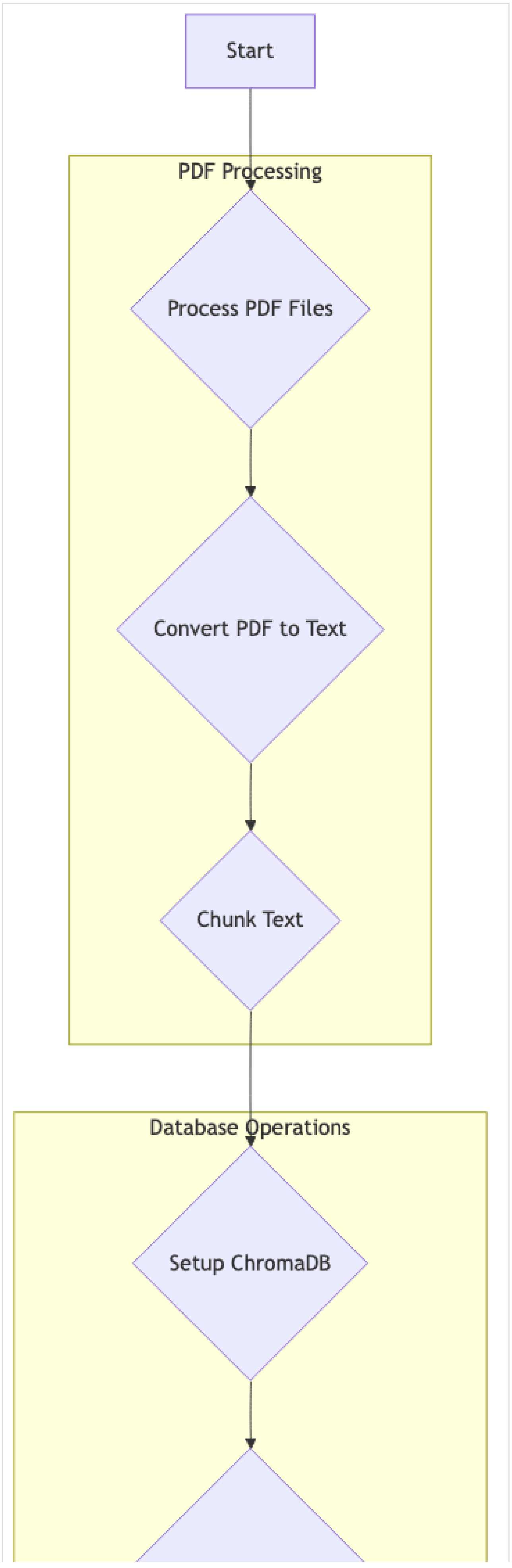
Purpose:

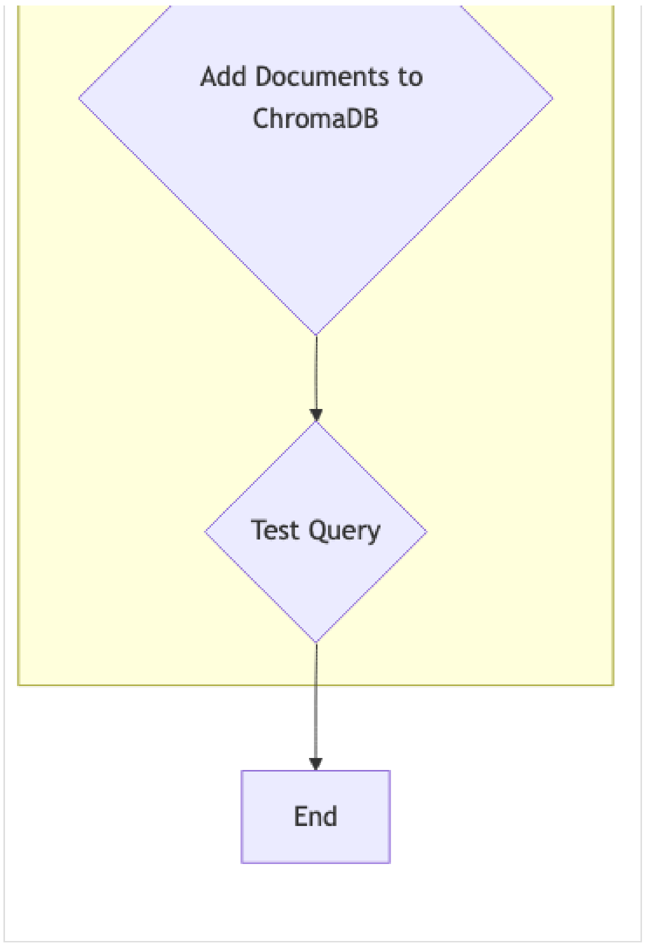
This script is responsible for creating a Retrieval Augmented Generation (RAG) database from a collection of PDF documents. It begins by locating all PDF files in a specified directory, then extracts the text content from each PDF. The extracted text is then broken down into smaller, overlapping chunks to optimize it for semantic search. These text chunks are then stored in a ChromaDB vector database, which creates vector embeddings for each chunk, enabling efficient similarity-based retrieval. Finally, the script runs a test query against the newly created database to verify that the process was successful and that the data is queryable.

Pseudocode:

```
START
    // RAG Database Creation Script for PDF Files
    // This script creates a RAG database from PDF files.

    // Configuration
    DEFINE PDF_DIR, RAG_DIR, TXT_DIR

    // Main Function
    FUNCTION main():
        PRINT "Cisco AI PDF RAG Database Creator" banner
        documents = process_pdf_files()
        chroma_client, collection = setup_chroma_db()
        add_documents_to_chroma(collection, documents)
        test_query(collection)
        PRINT "RAG database creation complete!"
    END FUNCTION

    // Helper Functions
    FUNCTION process_pdf_files():
        GET list of PDF files from PDF_DIR
        FOR each PDF file:
            text_content = convert_pdf_to_text(pdf_path)
            SAVE text_content to a .txt file
            chunks = chunk_text(text_content)
            ADD chunks to processed_documents list
        RETURN processed_documents
    END FUNCTION

    FUNCTION convert_pdf_to_text(pdf_path):
        READ PDF file
        EXTRACT text from each page
        RETURN extracted text
    END FUNCTION

    FUNCTION chunk_text(text, file_name):
        SPLIT text into overlapping chunks
        RETURN list of chunks
    END FUNCTION

    FUNCTION setup_chroma_db():
        INITIALIZE ChromaDB client
        CREATE a new collection
        RETURN chroma_client and collection
    END FUNCTION

    FUNCTION add_documents_to_chroma(collection, documents):
        PREPARE documents, ids, and metadatas
        ADD documents to the collection in batches
    END FUNCTION

    FUNCTION test_query(collection):
        DEFINE a test query
        QUERY the collection
        DISPLAY the results
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
PDF Processing
Process PDF Files
Convert PDF to Text
Chunk Text
Database Operations
Setup ChromaDB



a02_LLM_Access.py

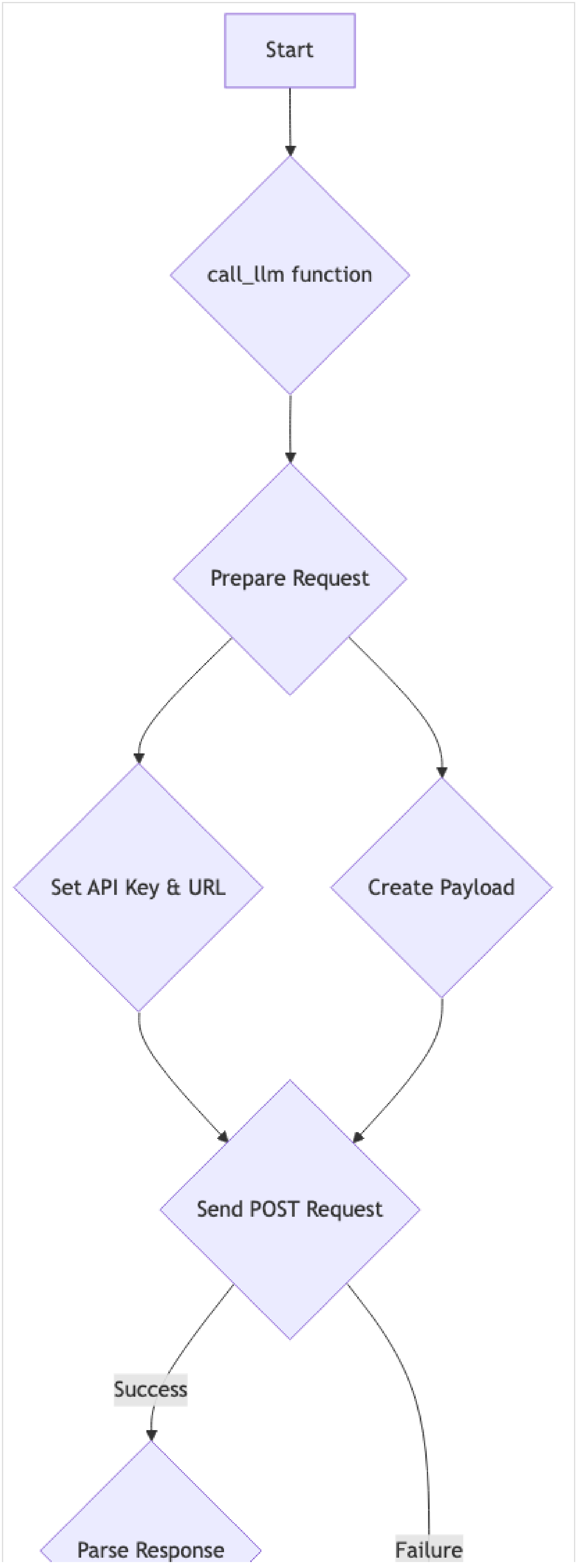
Purpose:

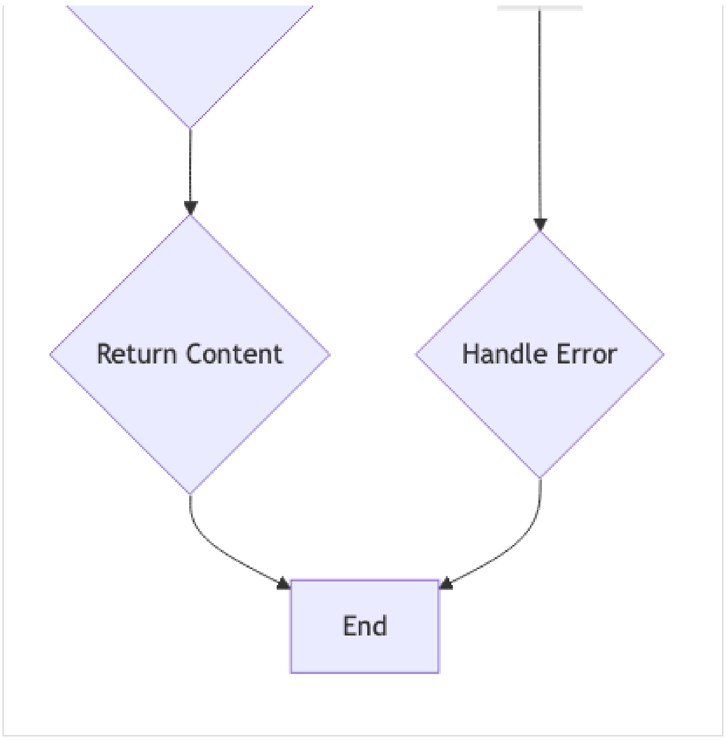
This script serves as a centralized module for accessing Large Language Models (LLMs) through the OpenRouter.ai API. It defines a single function, `call_llm`, which takes a user prompt and an optional system prompt as input. The function constructs a request payload with the specified model, temperature, and other parameters, then sends it to the OpenRouter API endpoint. It handles authentication by including a hardcoded API key in the request headers. The script includes error handling to manage failed API requests and will raise an exception if the request is unsuccessful. When run directly, it provides an example of how to use the `call_llm` function to ask a question and print the LLM's response.

Pseudocode:

```
START
    // LLM Access Module - OpenRouter.ai
    // This module provides a function to connect to OpenRouter.ai and access LLM models.

    // Main Function
    FUNCTION call_llm(prompt, system_prompt):
        DEFINE model, temperature, max_tokens
        GET API key
        DEFINE API URL and headers

        PREPARE messages array with system and user prompts
        PREPARE request payload (data)

        PRINT "Sending request to OpenRouter.ai"
        TRY
            POST request to API URL with headers and data
            RAISE exception for bad status codes
            PARSE JSON response
            RETURN response content
        CATCH Exception
            PRINT error message
            RAISE exception
        END TRY
    END FUNCTION

    // Example Usage
    IF script is run directly THEN
        response = call_llm("Explain how vector databases work...")
        PRINT response
    END IF
END
```

Mermaid Diagram:


Start
call_llm function
Prepare Request
Set API Key & URL
Create Payload
Send POST Request
Success
Failure
Parse Response



a04_CREATE_OUTLINE.py

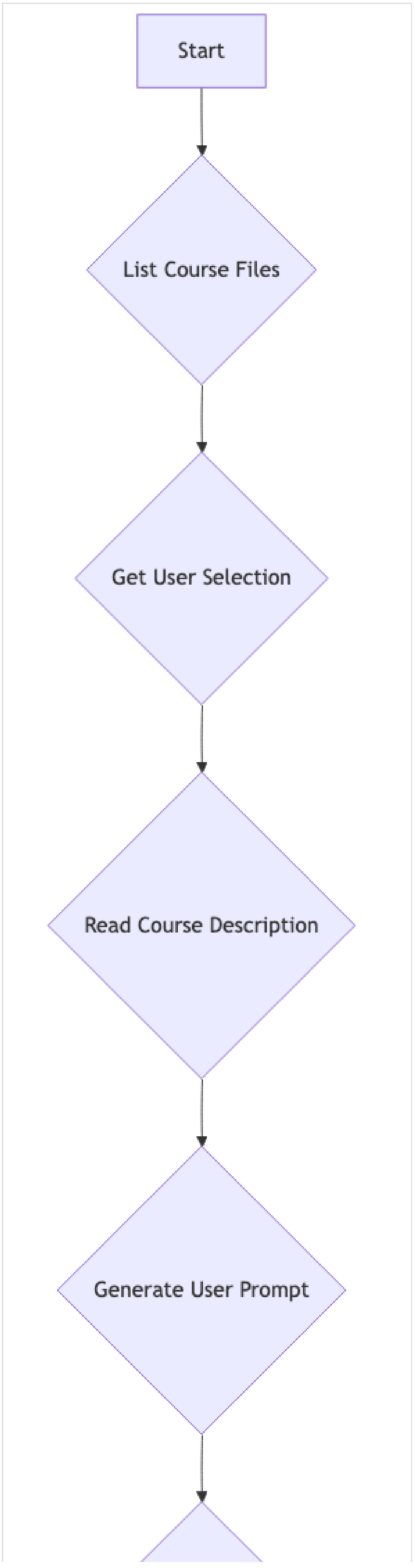
Purpose:

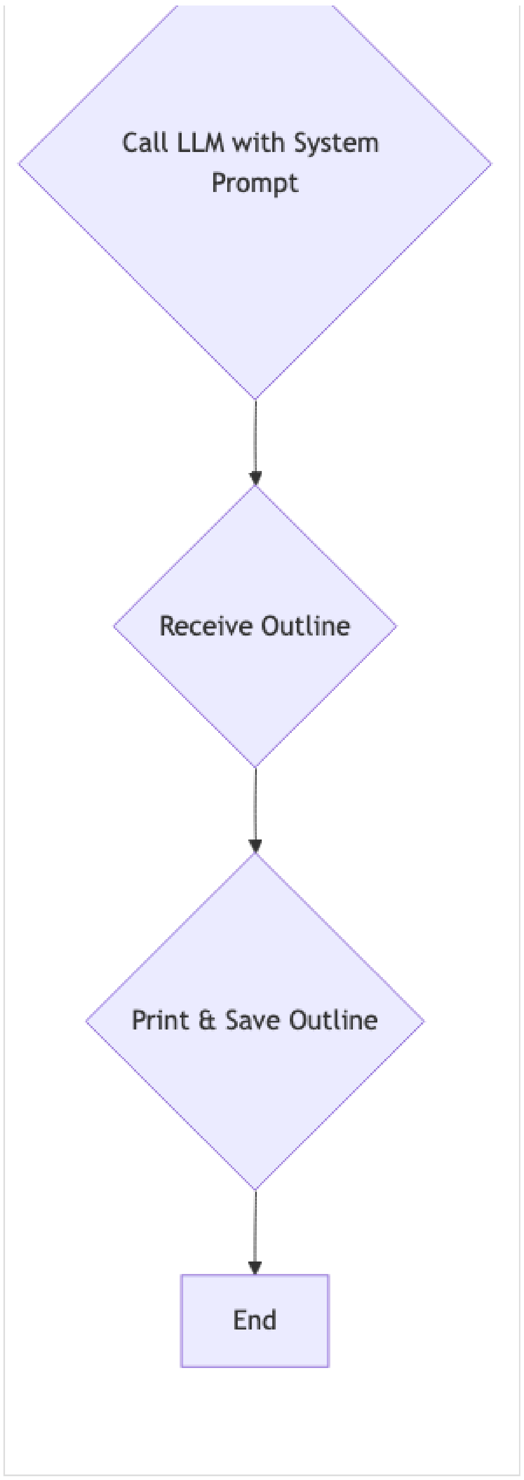
This script automates the creation of a detailed course outline from a high-level course description. It first presents the user with a list of available course description files and prompts them to select one. Once a file is chosen, the script reads its content and combines it with a specialized, hardcoded system prompt that instructs the LLM to act as an expert curriculum designer. This combined prompt is then sent to the LLM, which generates a comprehensive, well-structured course outline. The resulting outline is then printed to the console and saved to a text file for later use in the content generation pipeline.

Pseudocode:

```
START
    // Course Outline Generator
    // This script generates a course outline from a course description file.

    // Initialization
    IMPORT call_llm function
    DEFINE a static SYSTEM_PROMPT for curriculum design

    // Main Function
    FUNCTION main():
        PRINT "COURSE OUTLINE GENERATOR" banner
        GET list of course description files
        PROMPT user to select a file
        READ the selected file content
        CREATE user_prompt with the course description
        CALL the LLM with the system and user prompts
        PRINT the generated outline

        SAVE the outline to "course_outline.txt"
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
List Course Files
Get User Selection
Read Course Description
Generate User Prompt



a06-Student_Notes_Student_Handbook.py

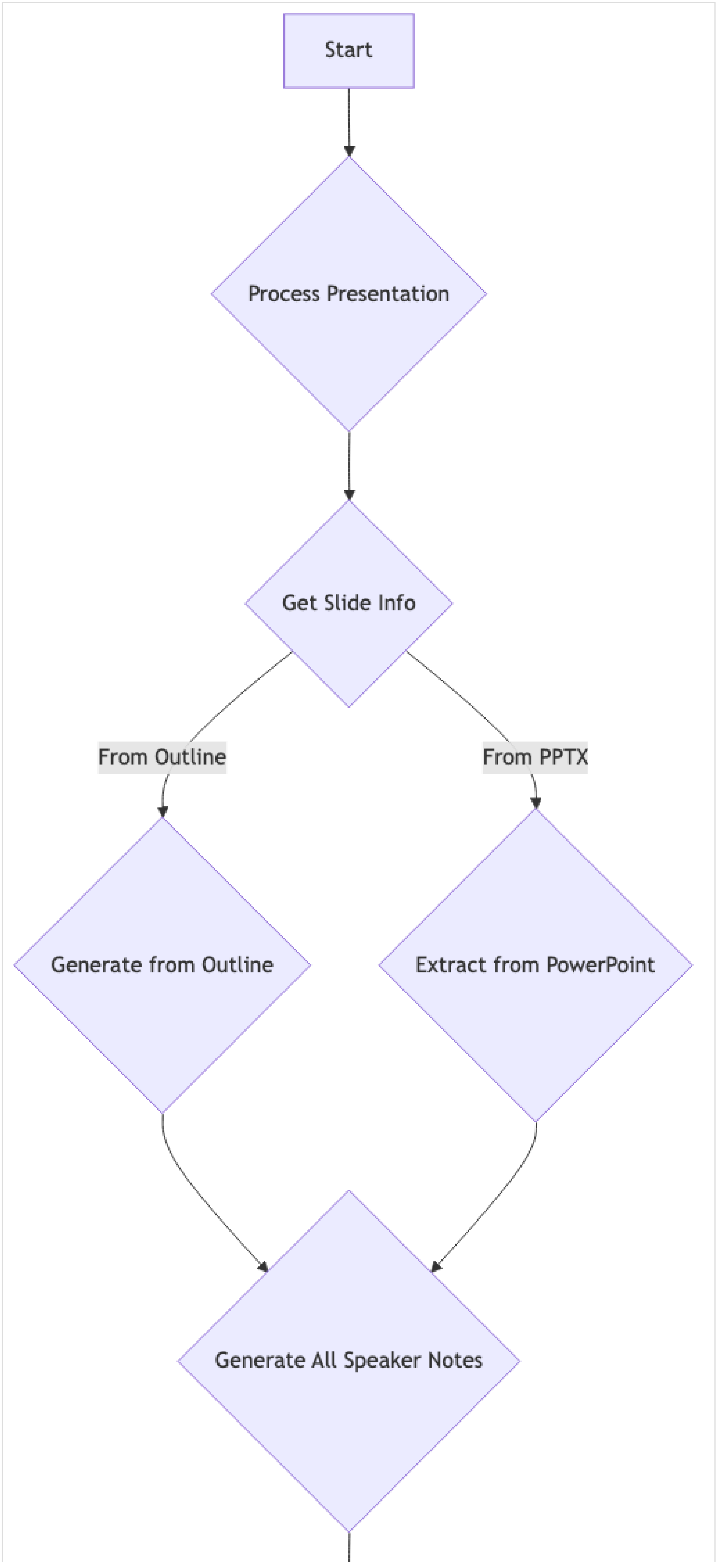
Purpose:

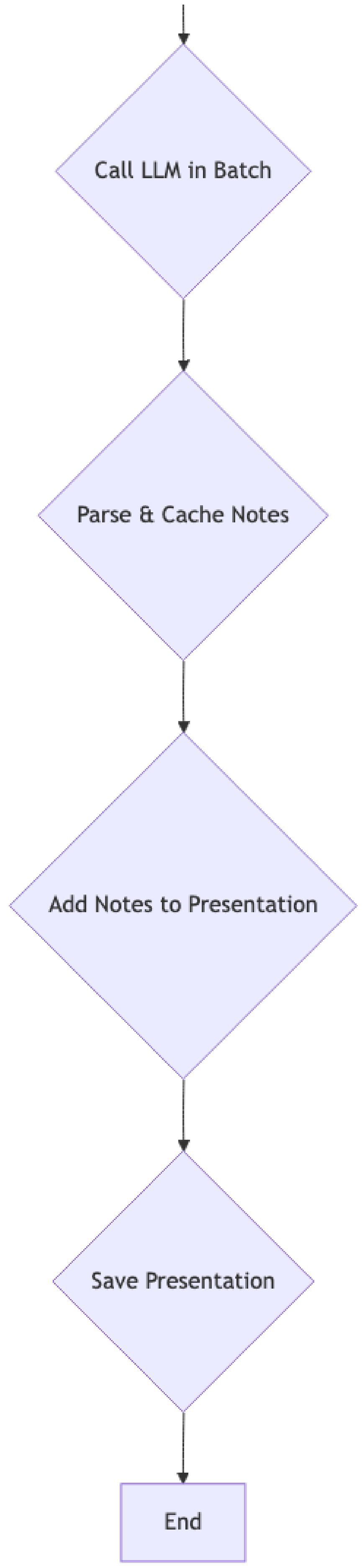
This script is designed to enrich a PowerPoint presentation by adding detailed speaker notes to each slide. It can either work from a pre-existing course outline or, if no outline is provided, it can extract the title and content directly from the slides of a PowerPoint file. The script then compiles the content of all slides into a single batch and sends it to a Large Language Model (LLM) with a prompt specifically designed to generate comprehensive speaker notes. Once the notes are generated, the script iterates through the presentation, matches each slide with its corresponding notes, and adds them to the notes section of the slide. This automated process ensures that every slide is accompanied by relevant, high-quality notes for the presenter.

Pseudocode:

```
START
    // Speaker Notes Generation Module
    // This module generates speaker notes for PowerPoint slides.

    // Initialization
    IMPORT necessary libraries and custom modules

    // Main Function
    FUNCTION process_presentation_with_notes(outline_data, pptx_file, max_slides):
        IF outline_data is not provided THEN
            EXTRACT slide content directly from the PowerPoint file
        ELSE
            GENERATE slides_info from the outline_data
        END IF

        all_notes = generate_all_speaker_notes(slides_info, max_slides)
        IF all_notes is not empty THEN
            add_speaker_notes_to_presentation(pptx_file, all_notes)
            RETURN True
        ELSE
            RETURN False
        END IF
    END FUNCTION

    // Helper Functions
    FUNCTION generate_all_speaker_notes(slides_info, max_slides):
        PREPARE a single batch prompt for all slides
        CALL the LLM to generate speaker notes for all slides
        PARSE the JSON response and cache the notes
        RETURN dictionary of slide titles to speaker notes
    END FUNCTION

    FUNCTION add_speaker_notes_to_presentation(pptx_file, slide_notes_dict):
        LOAD the presentation
        FOR each slide:
            FIND the slide title
            IF notes exist for the title THEN
                ADD the notes to the slide's notes page
            END IF
        END FOR
        SAVE the presentation
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        PARSE the outline file
        process_presentation_with_notes()
    END IF
END
```

Mermaid Diagram:


Start
Process Presentation
Get Slide Info
From Outline
From PPTX
Generate from Outline
Extract from PowerPoint
Generate All Speaker Notes


Call LLM in Batch
Parse & Cache Notes
Add Notes to Presentation
Save Presentation
End

a07_QUIZ_Per_Module.py

Purpose:

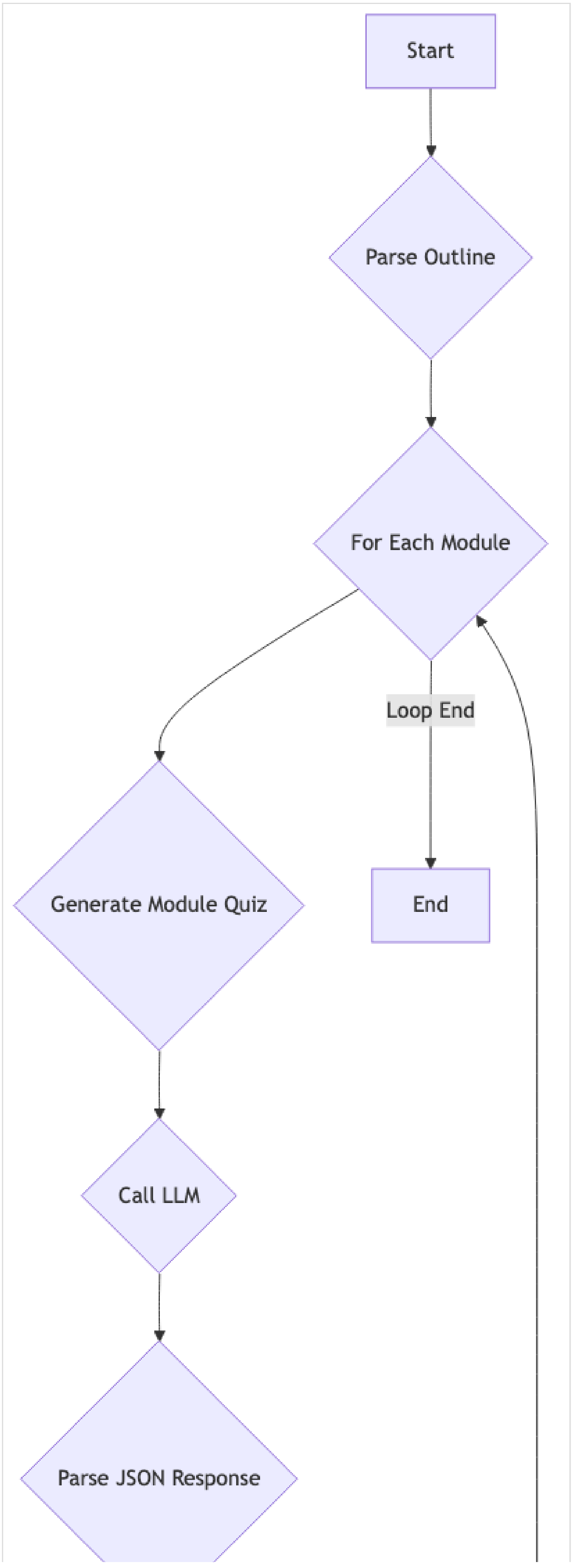
This script automates the creation of a multiple-choice quiz for each module of a course. It begins by parsing the course outline to identify the individual modules and their content. For each module, it sends the content to a Large Language Model (LLM) with a prompt that instructs it to generate a 10-question quiz. The LLM returns the quiz in a structured JSON format, which the script then parses to create two separate files: one for the quiz itself and another for the answer key. This process is repeated for every module, resulting in a complete set of quizzes that can be used to assess understanding of the course material.

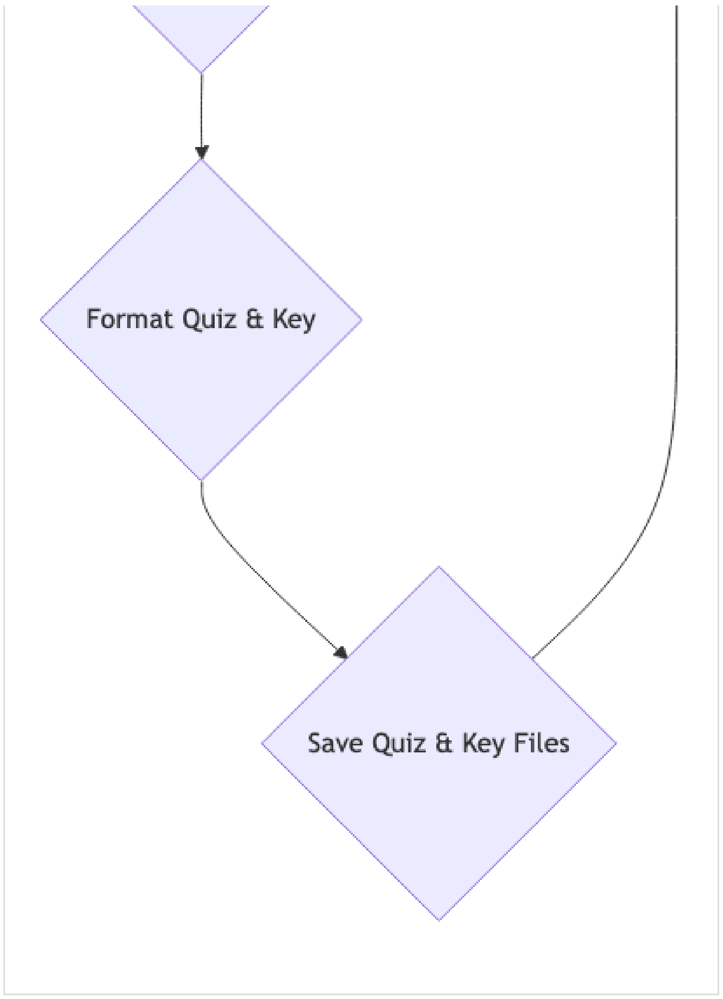
Pseudocode:

```
START
    // Quiz Generator for Course Modules
    // This script generates a quiz for each module in the course.

    // Initialization
    IMPORT necessary libraries and custom modules

    // Main Function
    FUNCTION main():
        PRINT "Course Quiz Generator" banner
        DETERMINE file paths
        PARSE the course outline file
        create_quiz_files(outline, course_title)
    END FUNCTION

    // Helper Functions
    FUNCTION create_quiz_files(outline, course_title):
        FOR each module in the outline:
            quiz_content, answer_key = generate_module_quiz(module_data, module_number)
            IF quiz and answer key are generated THEN
                SAVE the quiz to a file
                SAVE the answer key to a file
            END IF
        END FOR
    END FUNCTION

    FUNCTION generate_module_quiz(module_data, module_number):
        FORMAT the module content for the LLM prompt
        CONSTRUCT a system prompt for quiz creation
        CALL the LLM to generate the quiz
        PARSE the JSON response
        FORMAT the quiz and answer key
        RETURN the formatted quiz and answer key
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
```

```
            main()
        END IF
    END
```

Mermaid Diagram:


Start
Parse Outline
For Each Module
Loop End
Generate Module Quiz
End
Call LLM
Parse JSON Response



a08_Final_Exam.py

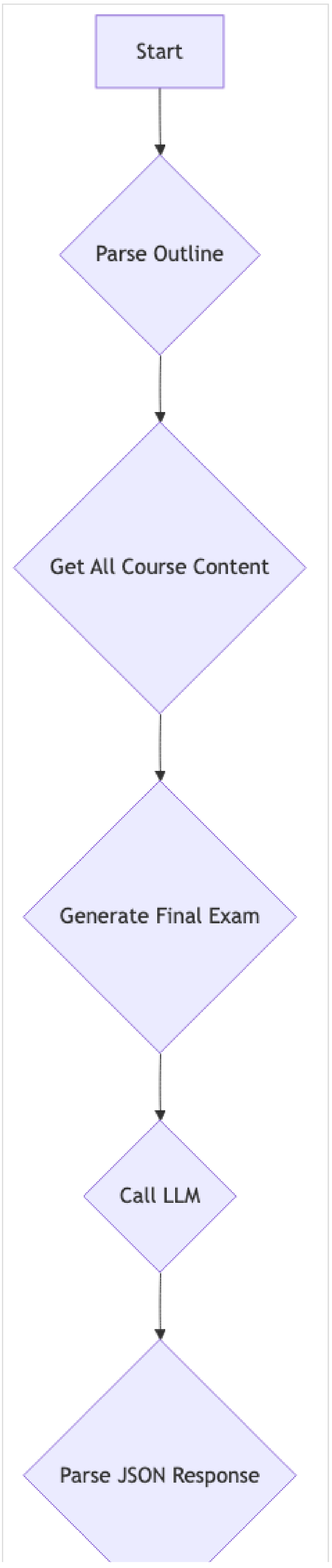
Purpose:

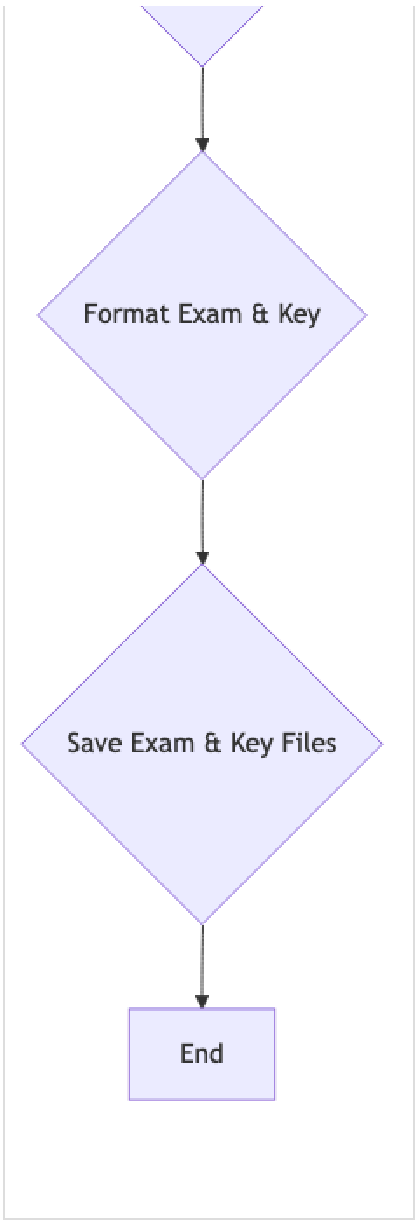
This script is designed to generate a comprehensive final exam for an entire course. It begins by parsing the course outline to gather all of the content from every module. This complete set of content is then sent to a Large Language Model (LLM) with a prompt that instructs it to create a 50-question multiple-choice exam that covers the material from all modules. The LLM returns the exam in a structured JSON format, which the script then uses to create two separate files: one for the final exam and another for the corresponding answer key. This provides a ready-to-use final assessment for the course.

Pseudocode:

```
START
    // Final Exam Generator for Course
    // This script generates a final exam for the entire course.

    // Initialization
    IMPORT necessary libraries and custom modules

    // Main Function
    FUNCTION main():
        PRINT "Course Final Exam Generator" banner
        DETERMINE file paths
        PARSE the course outline file to get all content
        exam_content, answer_key = generate_final_exam(all_content, course_title, module_count)
        IF exam and answer key are generated THEN
            create_exam_files(exam_content, answer_key, course_title)
        END IF
    END FUNCTION

    // Helper Functions
    FUNCTION generate_final_exam(course_content, course_title, module_count):
        FORMAT the course content for the LLM prompt
        CONSTRUCT a system prompt for final exam creation
        CALL the LLM to generate the exam
        PARSE the JSON response
        FORMAT the exam and answer key
        RETURN the formatted exam and answer key
    END FUNCTION

    FUNCTION create_exam_files(exam_content, answer_key, course_title):
        SAVE the exam to a file
        SAVE the answer key to a file
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
Parse Outline
Get All Course Content
Generate Final Exam
Call LLM
Parse JSON Response



PowerPoint & Asset Creation

a05_CREATE_POWERPOINT.py

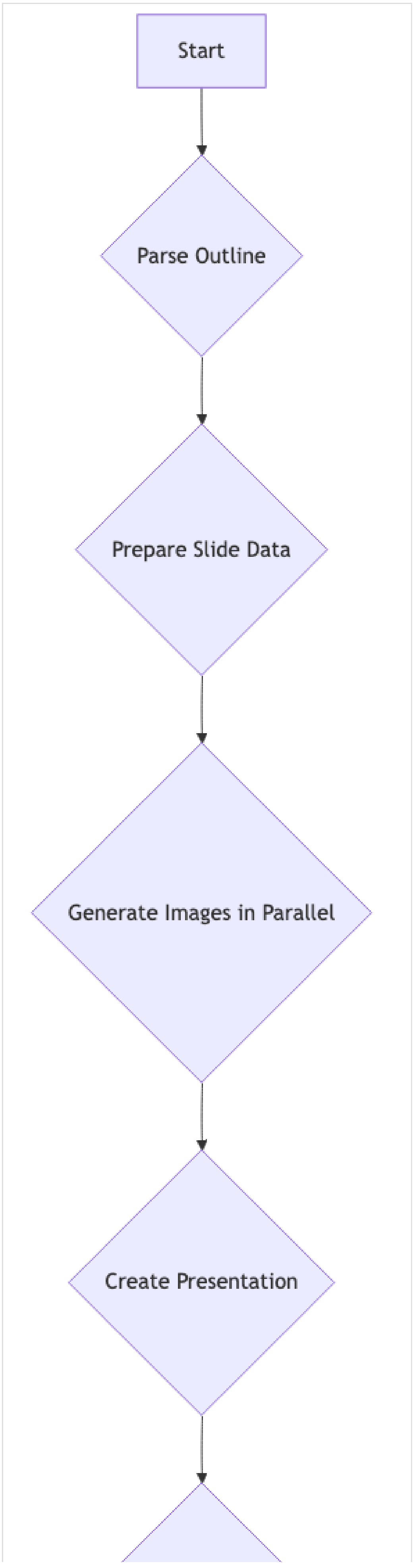
Purpose:

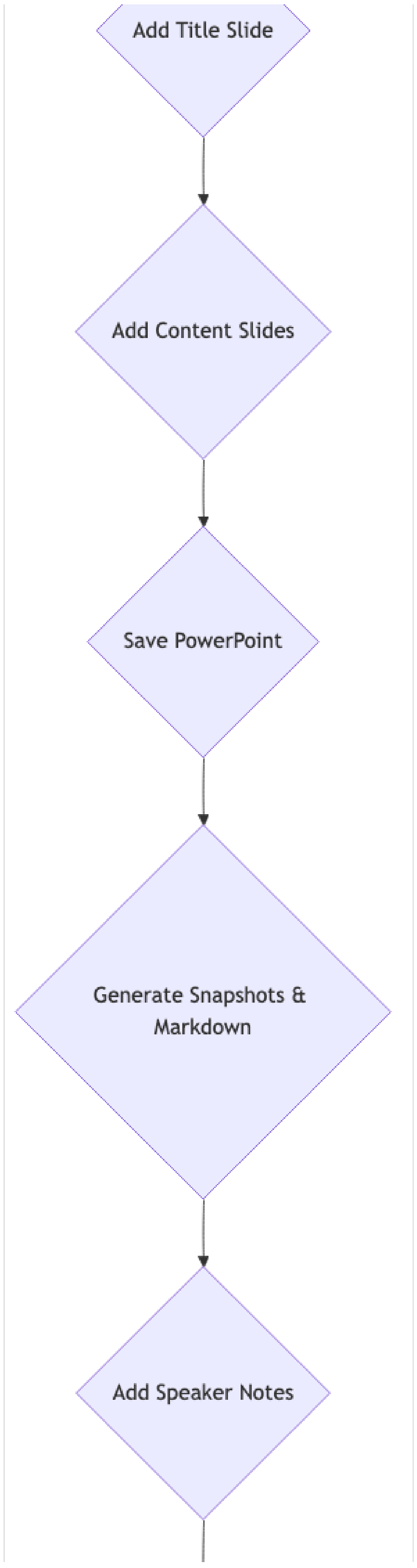
This script is the core of the presentation generation process, responsible for converting a text-based course outline into a complete PowerPoint presentation. It begins by parsing the outline file to understand the structure of the course, including modules, topics, and subtopics. For each of these components, it prepares the data for a corresponding slide. A key feature of this script is its ability to generate images for each slide in parallel, which significantly speeds up the process. It then creates a new presentation, adds a title slide, and then iterates through the prepared slide data to create content slides, each with its own title, bullet points, and a pre-generated, contextually relevant image. Finally, it saves the completed presentation, generates a Markdown version with slide snapshots, and adds speaker notes to the PowerPoint file.

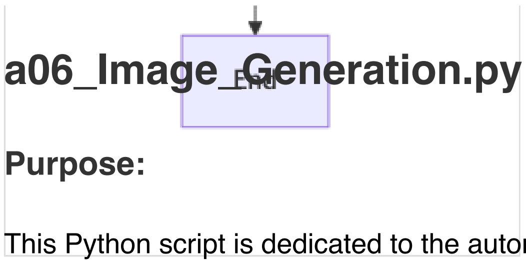
Pseudocode:

```
START
    // PowerPoint Generator from Course Outline
    // This script creates a PowerPoint presentation from a course outline.

    // Initialization
    IMPORT necessary libraries and custom modules
    DEFINE constants for file paths and slide generation

    // Main Function
    FUNCTION main():
        PRINT "Course PowerPoint Generator" banner
        DETERMINE file paths based on current directory
        CHECK for a template file
        PARSE the course outline file

        PREPARE slide data for all slides
        GENERATE all slide images in parallel
        CREATE a new presentation or use a template
        ADD a title slide
        ADD content slides with pre-generated images
        SAVE the presentation

        GENERATE slide snapshots for Markdown export
        GENERATE and save a Markdown version of the presentation
        ADD speaker notes to the presentation
    END FUNCTION

    // Helper Functions
    FUNCTION parse_outline(file_path):
        READ and PARSE the outline file
        RETURN a nested dictionary and the course title
    END FUNCTION

    FUNCTION prepare_slides_data(outline_data, max_slides):
        PREPARE data for each slide (title, content, color)
        RETURN list of slide data dictionaries
    END FUNCTION

    FUNCTION generate_slide_images_parallel(slides_data, batch_size):
        GENERATE all enhanced prompts in a single batch
        GENERATE images from prompts in parallel batches
        RETURN dictionary of slide titles to image paths
    END FUNCTION

    FUNCTION add_slides_to_presentation(prs, slides_data, image_paths_by_title):
        FOR each slide_data:
            CREATE a content slide with the pre-generated image
        END FOR
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
Parse Outline
Prepare Slide Data
Generate Images in Parallel
Create Presentation


Add Title Slide
Add Content Slides
Save PowerPoint
Generate Snapshots & Markdown
Add Speaker Notes



a06_Image_Generation.py

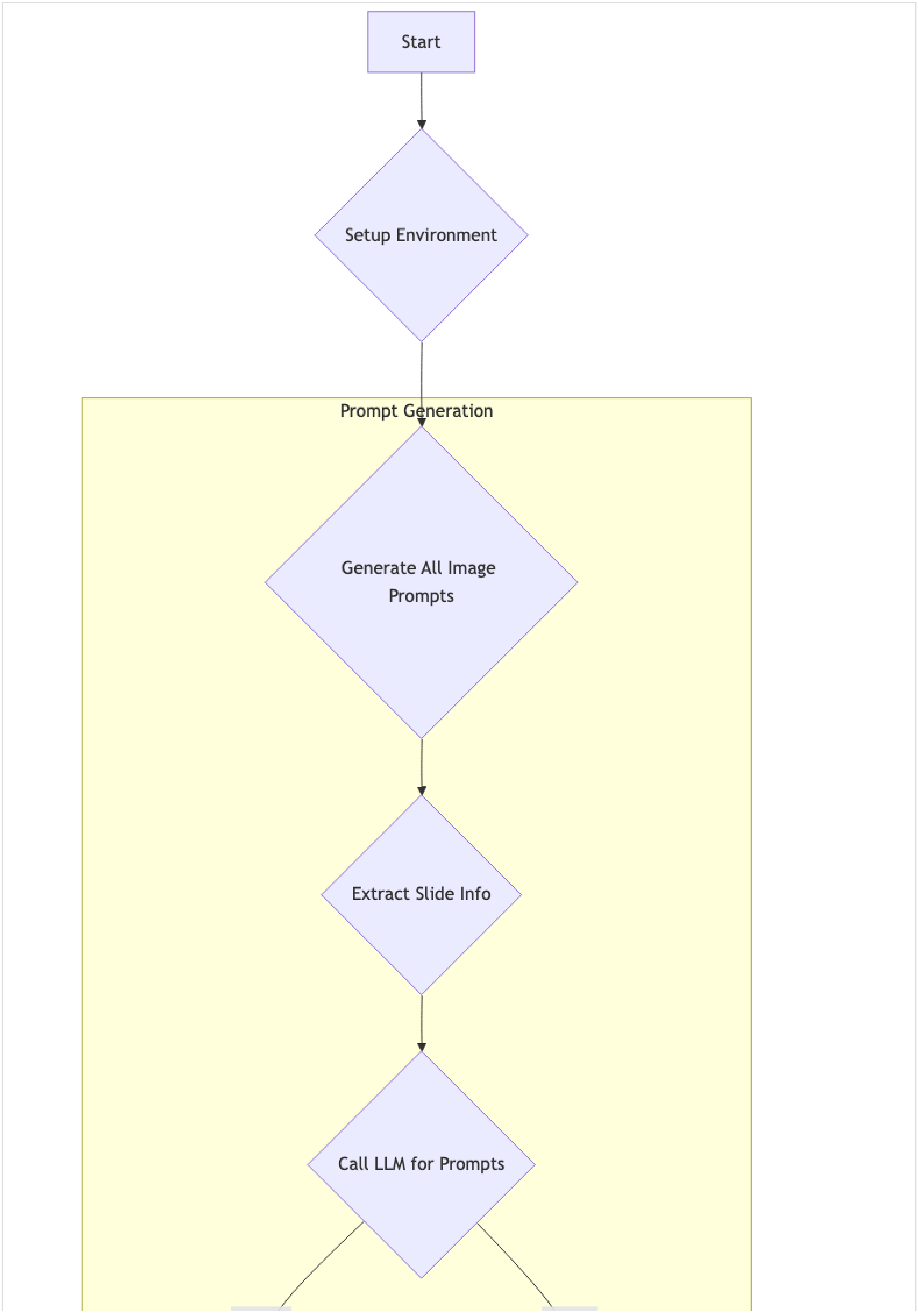
Purpose:

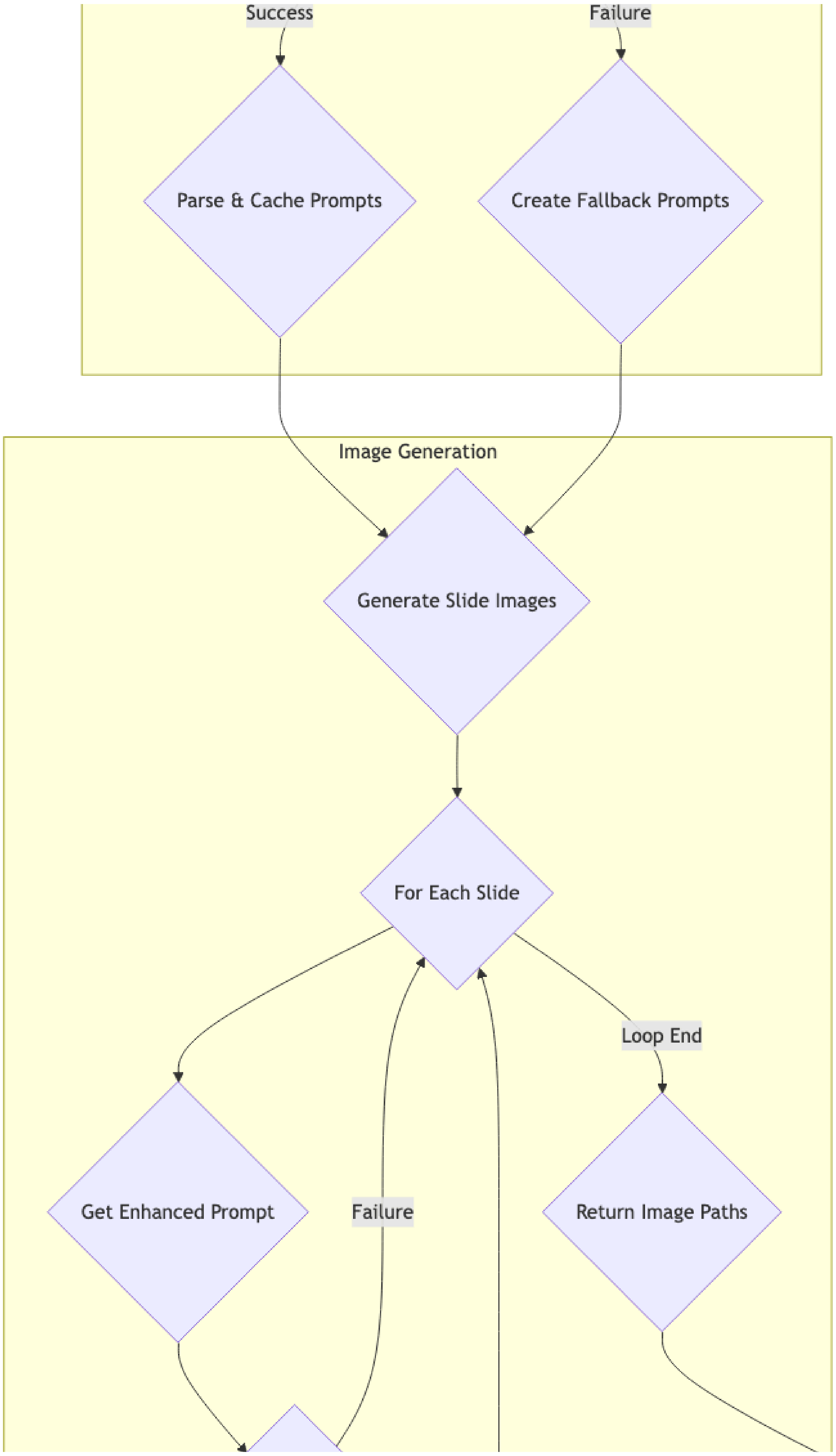
This Python script is dedicated to the automated generation of images for a presentation. Its primary role is to create visually compelling images for each slide, based on the slide's content. The script is designed to be modular, separating the image generation logic from the main presentation creation process. The script begins by setting up its environment, which includes dynamically importing a Large Language Model (LLM) module for generating creative image prompts. It then reads the presentation outline to extract the title and content of each slide. For each slide, it generates a detailed and context-aware prompt for an image generation AI. This is a key feature, as it leverages the LLM to create prompts that are more descriptive and artistic than simple slide titles, leading to higher quality images. Once the prompts are generated, the script proceeds to generate the images themselves. It calls an image generation API for each prompt, creating a unique image for every slide. The script includes robust error handling, with fallback mechanisms to create basic prompts if the LLM fails. It also manages the file paths and saves the generated images to a designated directory, ensuring they are organized and ready to be embedded into the final presentation. The script is designed to be run as part of a larger workflow, but can also be executed independently for testing purposes.

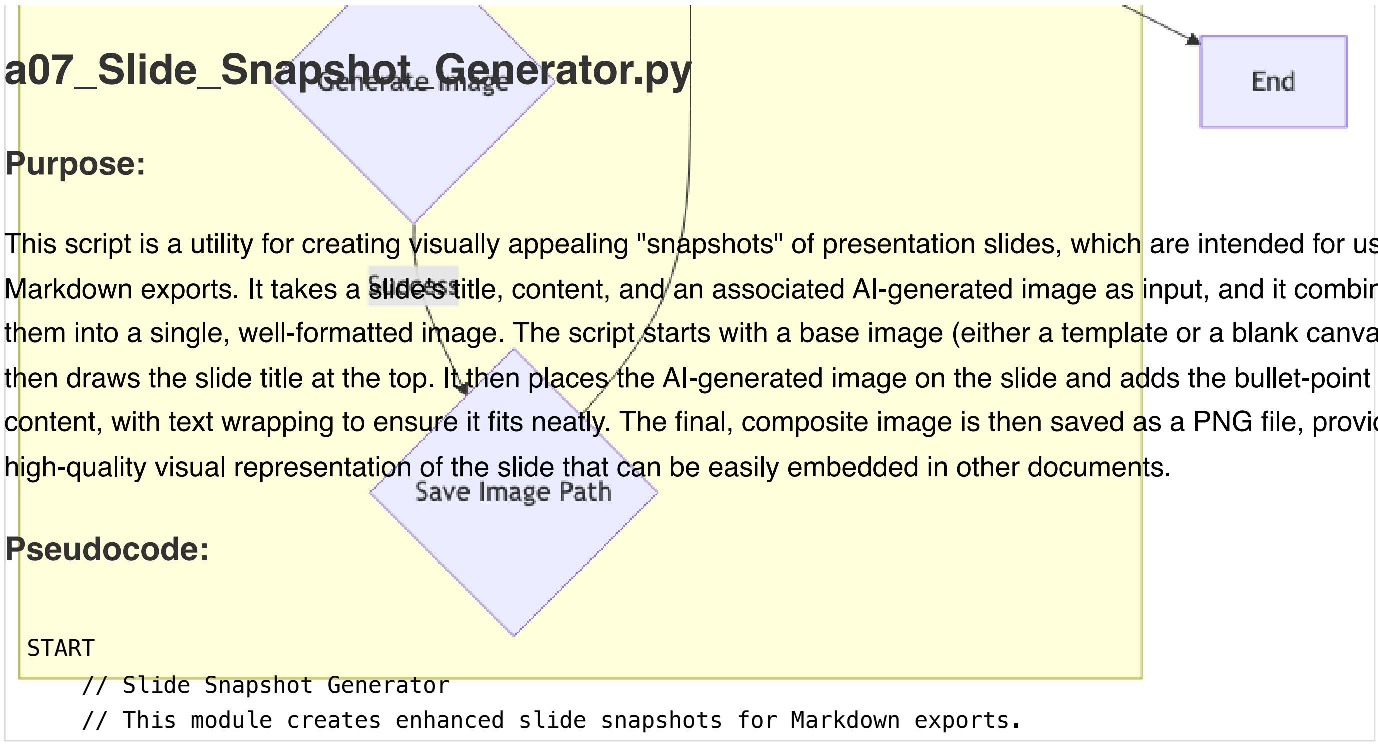
Pseudocode:

```
START
    // Image Generation Module for Course Presentations
    // This module handles image prompt generation and image creation for presentation slides.

    // Configuration
    IMAGE_WIDTH = 512
    IMAGE_HEIGHT = 1024
    DEFAULT_IMAGE_DIR = "slide_images"
    config = {
        "image_dir": DEFAULT_IMAGE_DIR,
        "prompt_style": "professional business",
        "focus": "people and technology concepts with an exciting and modern design",
        "color_scheme": "modern corporate look"
    }

    // Initialization
    FUNCTION setup_environment():
        IMPORT LLM module dynamically
        SETUP output directory
        CREATE image directory if it doesn't exist
    END FUNCTION

    FUNCTION get_current_directory():
        READ "current_directory.txt"
        RETURN directory path
    END FUNCTION

    // Core Logic
    FUNCTION generate_all_image_prompts(outline_data, max_slides):
        PRINT "Preparing slide content for batch prompt generation..."
        slides_info = extract_slide_info(outline_data, max_slides)

        system_prompt = "You are an expert AI image prompt engineer..."
        user_prompt = "Generate unique image prompts for each of these presentation slides: [slides_json]"

        TRY
            enhanced_prompts_json = call_llm(user_prompt, system_prompt)
            save_prompts_to_file(enhanced_prompts_json)
            prompt_cache = parse_json(enhanced_prompts_json)
        CATCH Exception
            PRINT "Error generating enhanced prompts"
            prompt_cache = create_fallback_prompts(slides_info)
        END TRY

        RETURN prompt_cache
    END FUNCTION

    FUNCTION generate_slide_images_parallel(slides_data, batch_size):
        PRINT "Preparing to generate images for slides"
        image_paths_by_title = {}
        existing_files = get_existing_images()

        FOR each slide in slides_data:
            image_filename = create_numeric_filename(slide_number)
            output_path = image_dir + image_filename

            IF image already exists THEN
                PRINT "Image already exists"
            ELSE
                prompt = get_enhanced_prompt(slide["title"], slide["content"])
                PRINT "Generating image for slide"
                TRY
                    generate_image(prompt, output_path)
                CATCH Exception
                    PRINT "Error generating image"
                END TRY
            END IF

            IF image exists THEN
                image_paths_by_title[slide["title"]] = output_path
            END IF
        END FOR

        RETURN image_paths_by_title
    END FUNCTION

    // Helper Functions
    FUNCTION extract_slide_info(outline_data, max_slides):
        EXTRACT slide titles and content from the outline
        RETURN list of slide dictionaries
    END FUNCTION

    FUNCTION get_enhanced_prompt(slide_title, slide_content):
        IF slide_title in prompt_cache THEN
            RETURN prompt_cache[slide_title]
        ELSE
            RETURN create_fallback_prompt(slide_title)
        END IF
    END FUNCTION

    FUNCTION generate_image_for_slide(slide_title, slide_content):
        prompt = get_enhanced_prompt(slide_title, slide_content)
        safe_title = sanitize_filename(slide_title)
        image_path = image_dir + "slide_" + safe_title + ".png"

        TRY
            image_files = generate_image(prompt, image_path)
            RETURN first image path
        CATCH Exception
            PRINT "Error generating image"
            RETURN None
        END TRY
    END FUNCTION

    // Main Execution
    IF script is run directly THEN
        main()
    END IF

    setup_environment()
END
```

Mermaid Diagram:


Start
Setup Environment
Prompt Generation
Generate All Image Prompts
Extract Slide Info
Call LLM for Prompts


Success
Failure
Parse & Cache Prompts
Create Fallback Prompts
Image Generation
Generate Slide Images
For Each Slide
Loop End
Get Enhanced Prompt
Failure
Return Image Paths



a07_Slide_Snapshot_Generator.py

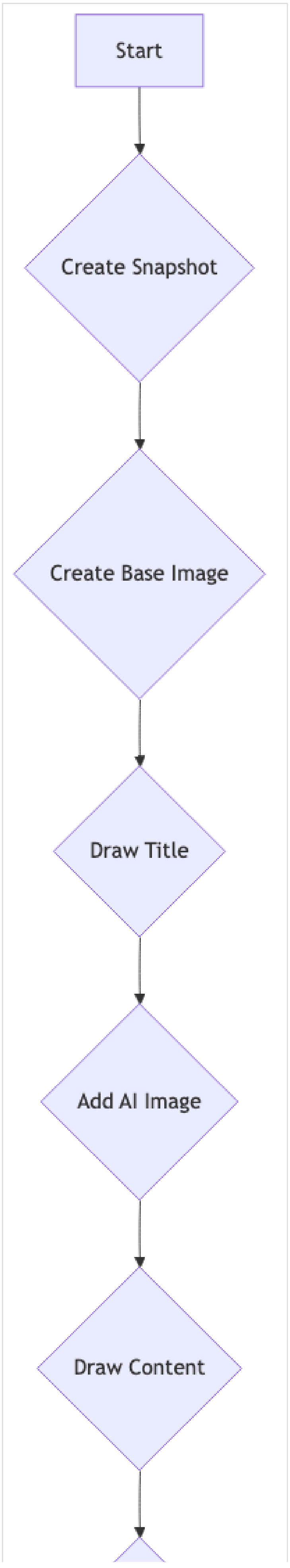
Purpose:

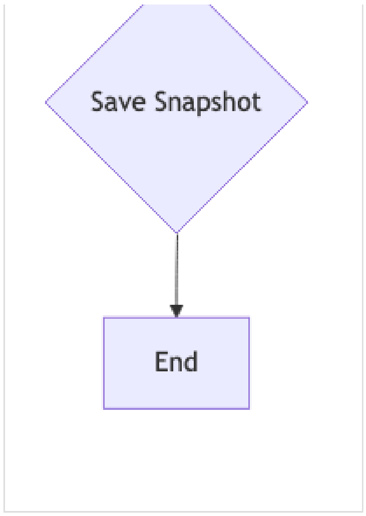
This script is a utility for creating visually appealing "snapshots" of presentation slides, which are intended for use in Markdown exports. It takes a slide's title, content, and an associated AI-generated image as input, and it combines them into a single, well-formatted image. The script starts with a base image (either a template or a blank canvas), then draws the slide title at the top. It then places the AI-generated image on the slide and adds the bullet-point content, with text wrapping to ensure it fits neatly. The final, composite image is then saved as a PNG file, providing a high-quality visual representation of the slide that can be easily embedded in other documents.

Pseudocode:

```
START
    // Slide Snapshot Generator
    // This module creates enhanced slide snapshots for Markdown exports.

    // Constants
    DEFINE image dimensions, font sizes, colors, and padding

    // Main Function
    FUNCTION create_slide_snapshot(title, content, ai_image_path, output_path, template_path):
        CREATE a base image (from template or blank)
        CREATE a drawing context
        LOAD fonts
        DRAW the title at the top

        IF an AI-generated image is available THEN
            RESIZE and PASTE the image onto the slide
        END IF

        DRAW the content bullet points with text wrapping
        SAVE the resulting image
        RETURN the path to the snapshot
    END FUNCTION

    FUNCTION generate_snapshots_for_presentation(slides_data, ai_image_paths, output_dir, template_path):
        FOR each slide_data:
            GET the AI image path
            GENERATE a safe filename
            create_slide_snapshot()
            ADD the snapshot path to a dictionary
        END FOR
        RETURN the dictionary of snapshot paths
    END FUNCTION

    // Example Usage
    IF script is run directly THEN
        create_slide_snapshot() for a test case
    END IF
END
```

Mermaid Diagram:


Start
Create Snapshot
Create Base Image
Draw Title
Add AI Image
Draw Content



a10_Audio_Generation_for_Slides.py

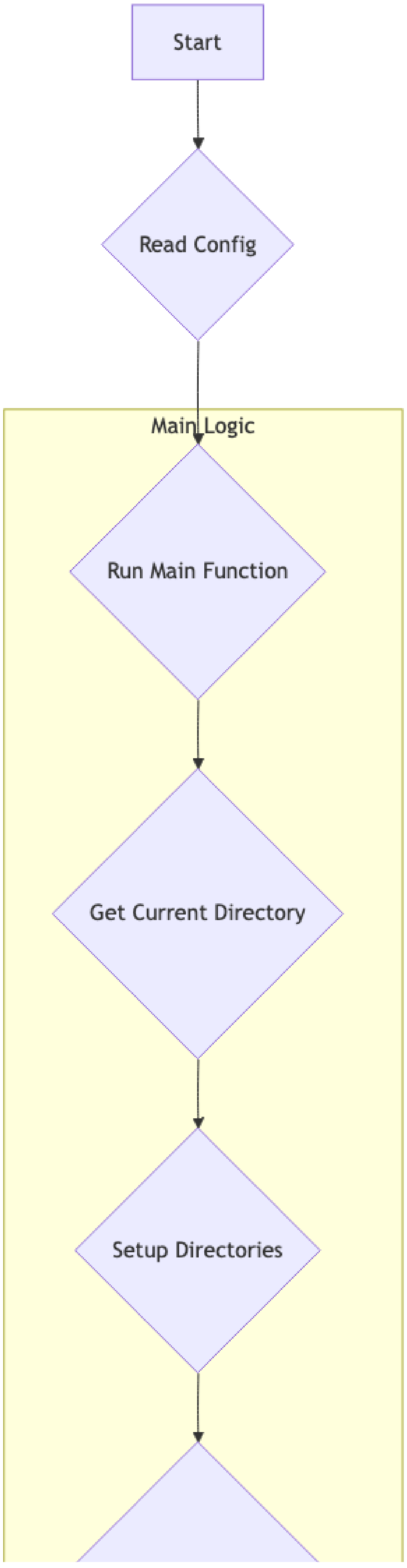
Purpose:

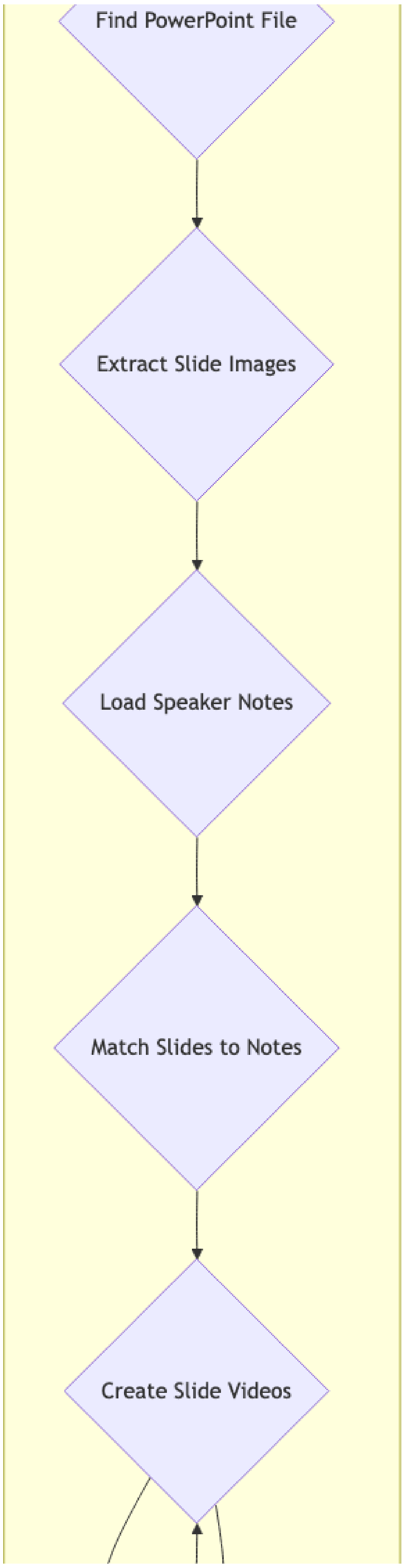
This Python script is designed to automate the conversion of a PowerPoint presentation into a video with audio narration. The script orchestrates a multi-step process that begins by identifying the presentation file and associated speaker notes. It then extracts each slide as an image and aligns it with the corresponding narration text. The core functionality of the script revolves around its ability to generate high-quality audio for each slide. It primarily utilizes the Google Cloud Text-to-Speech API to synthesize voiceovers from the speaker notes, but also includes robust fallback mechanisms. If the Google Cloud service is unavailable or fails, the script can revert to using local text-to-speech engines like `pyttsx3` or the native `say` command on macOS, ensuring that the process can complete even without cloud connectivity. Once the audio is generated for each slide, the script determines the appropriate duration for each slide based on the length of its audio track. It then combines the slide images and their corresponding audio files into individual video clips. Finally, it concatenates these clips into a single, cohesive video file, effectively transforming a static presentation into a dynamic, narrated video. The script also includes utility functions for file management, such as renaming slides to a standardized format, and dependency checking to ensure all required libraries are installed.

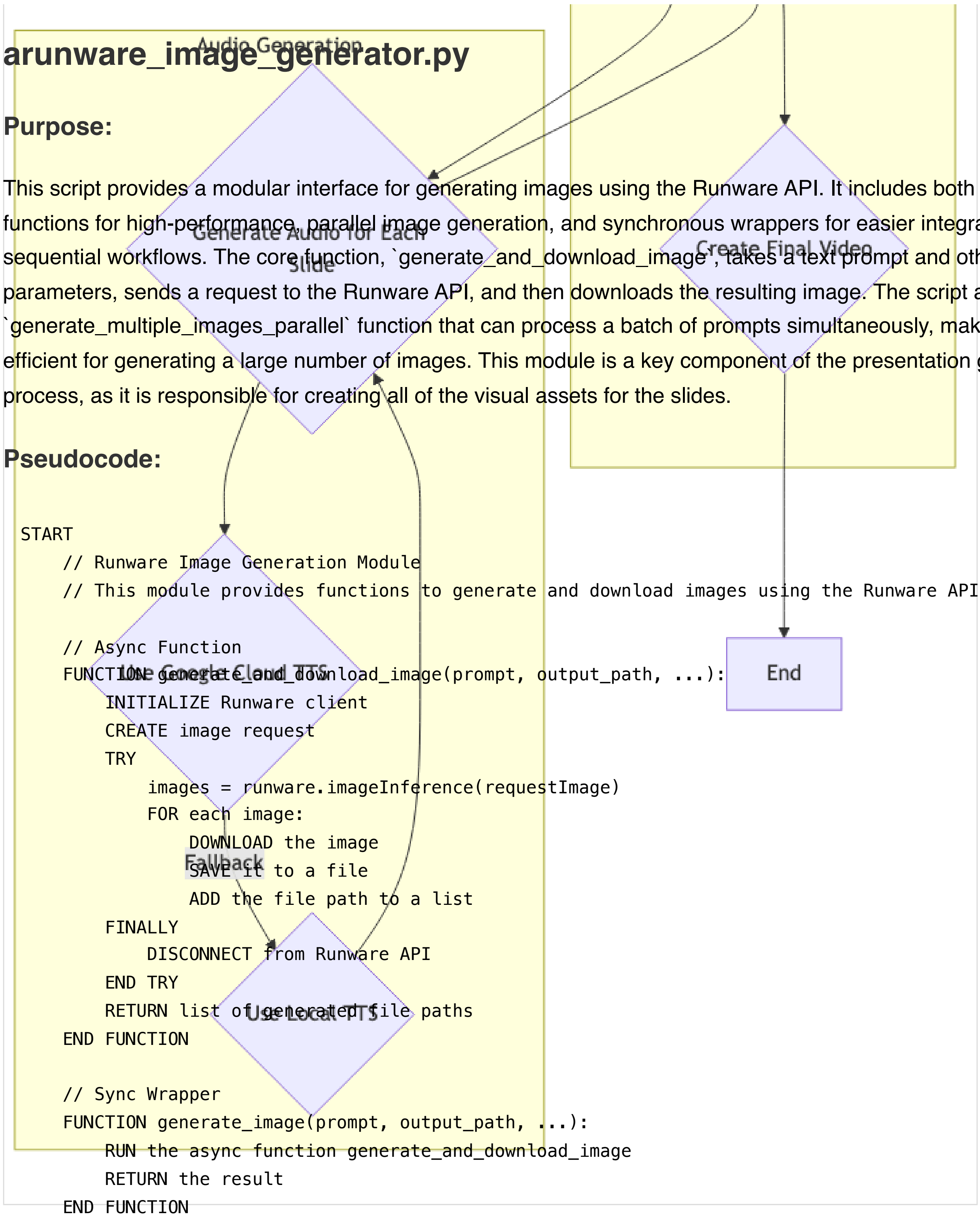
Pseudocode:

```
START
    // PowerPoint to Video Converter with Audio Narration
    // This script converts a PowerPoint presentation into a video with audio narration.

    // Configuration
    CURRENT_DIRECTORY_FILE = "current_directory.txt"
    ENHANCED_NOTES_FILE = "06_Enhanced_Notes.txt"
    OUTPUT_VIDEO_NAME = "course_video.mp4"
    GCP_SERVICE_ACCOUNT_FILE = "gcp-service-account.json"
    VOICE_NAME = "en-US-Neural2-F"
    VOICE_LANGUAGE_CODE = "en-US"

    // Main Function
    FUNCTION main(max_slides):
        Print "PowerPoint to Video Converter" banner

        current_dir = get_current_directory()
        paths = setup_directories(current_dir)

        pptx_file = find_pptx_file(paths["base"])
        IF pptx_file is not found THEN
            Print "Error: No PowerPoint file found"
            RETURN
        END IF

        notes_file = paths["base"] + ENHANCED_NOTES_FILE
        slide_image_paths = extract_slides_as_images(pptx_file, paths["images"])
        IF slide_image_paths is empty THEN
            RETURN
        END IF

        notes_data = load_speaker_notes(notes_file)
        IF notes_data is empty THEN
            Print "Warning: No speaker notes found. Creating silent video."
        END IF

        IF max_slides is set THEN
            limit slide_image_paths to max_slides
        END IF

        slide_matches = match_slides_to_notes(slide_image_paths, notes_data)
        slide_data = create_slide_videos(slide_matches, paths["audio"])
        IF slide_data is empty THEN
            Print "Error: Failed to create slide data."
            RETURN
        END IF

        output_video = paths["video"] + OUTPUT_VIDEO_NAME
        create_final_video(slide_data, output_video)
    END FUNCTION

    // Helper Functions
    FUNCTION get_current_directory():
        READ CURRENT_DIRECTORY_FILE
        RETURN directory path
    END FUNCTION

    FUNCTION setup_directories(base_dir):
        CREATE "slide_snapshots", "audio", "video" directories if they don't exist
        RETURN dictionary of paths
    END FUNCTION

    FUNCTION extract_slides_as_images(pptx_file, output_dir):
        FIND existing slide snapshots in output_dir
        SORT and RETURN list of image paths
    END FUNCTION

    FUNCTION load_speaker_notes(notes_file):
        READ and PARSE enhanced notes JSON file
        RETURN dictionary of notes
    END FUNCTION



    FUNCTION match_slides_to_notes(slide_image_paths, notes_data):
        MATCH slides to notes based on their order
        RETURN list of (slide_image_path, speaker_note) tuples
    END FUNCTION

    FUNCTION create_slide_videos(slide_matches, audio_dir):
        tts_client = setup_text_to_speech_client()
        FOR each slide_match:
            generate_audio_for_slide(note_text, audio_file, tts_client)
            determine_slide_duration from audio length
            create_video_clip for the slide
        RETURN list of video clips or data for ffmpeg
    END FUNCTION

    FUNCTION setup_text_to_speech_client():
        INITIALIZE Google Cloud Text-to-Speech client
        RETURN client
    END FUNCTION

    FUNCTION generate_audio_for_slide(text, output_file, tts_client):
        TRY to generate audio using Google Cloud TTS
        IF fails, FALLBACK to local TTS (pyttsx3 or macOS 'say')
        RETURN path to audio file
    END FUNCTION

    FUNCTION create_final_video(slide_data, output_path):
        IF using moviepy THEN
            CONCATENATE video clips
            WRITE final video file
        ELSE (using ffmpeg)
            CREATE individual video segments for each slide
            CONCATENATE segments using ffmpeg
        END IF
        RETURN path to final video
    END FUNCTION

    FUNCTION rename_slides(slides_dir):
        RENAME slide images to a standard format (e.g., "01-slide.png")
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        PARSE command-line arguments (--max-slides, --rename-slides)
        check_dependencies()
        IF --rename-slides is present THEN
            rename_slides()
        ELSE
            main()
        END IF
    END IF
END
```

Mermaid Diagram:


Start
Read Config
Main Logic
Run Main Function
Get Current Directory
Setup Directories


Find PowerPoint File
Extract Slide Images
Load Speaker Notes
Match Slides to Notes
Create Slide Videos



arunware_image_generator.py

Purpose:

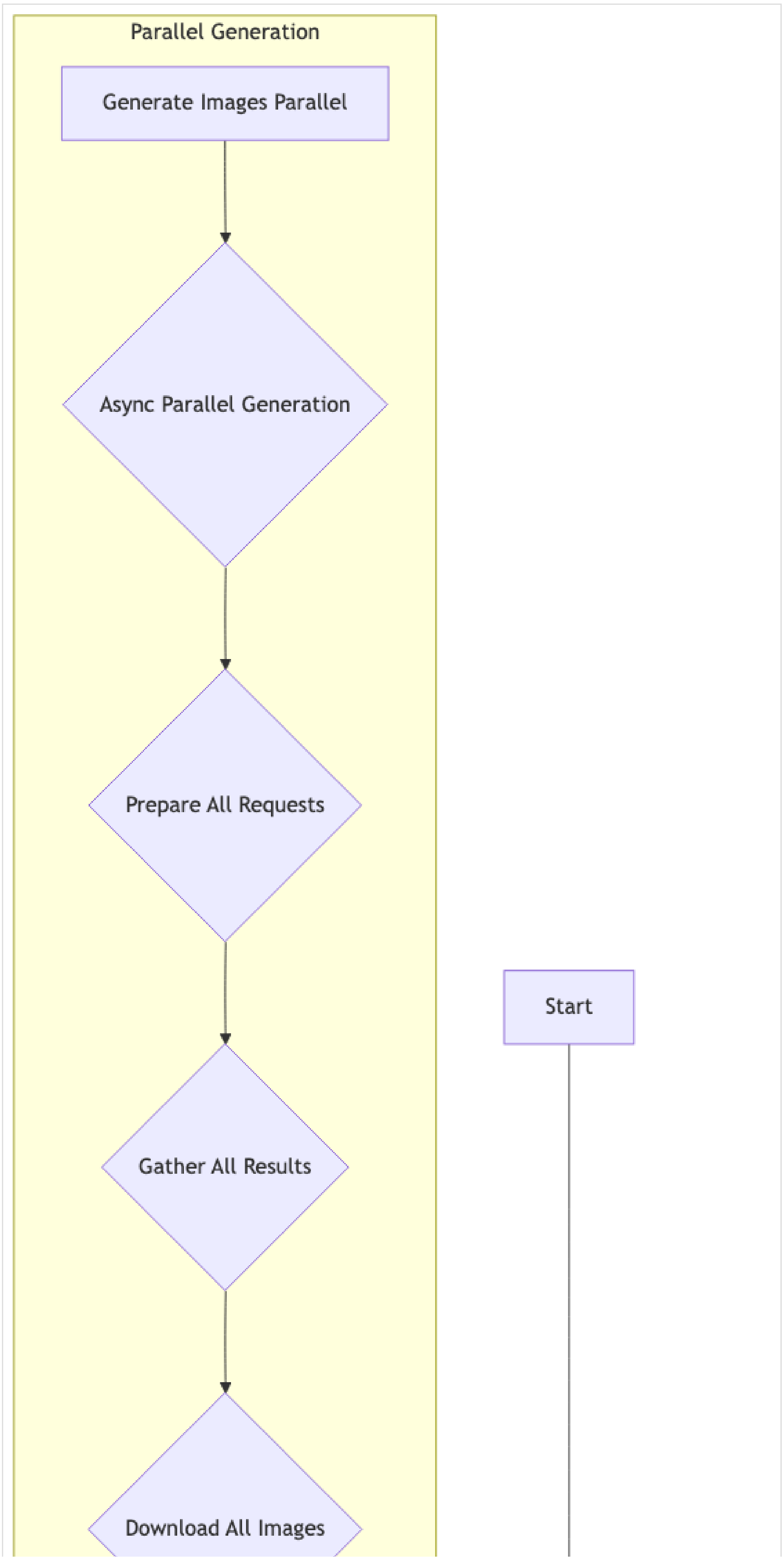
This script provides a modular interface for generating images using the Runware API. It includes both asynchronous functions for high-performance, parallel image generation, and synchronous wrappers for easier integration into sequential workflows. The core function, `generate_and_download_image`, takes a text prompt and other parameters, sends a request to the Runware API, and then downloads the resulting image. The script also offers a `generate_multiple_images_parallel` function that can process a batch of prompts simultaneously, making it highly efficient for generating a large number of images. This module is a key component of the presentation generation process, as it is responsible for creating all of the visual assets for the slides.

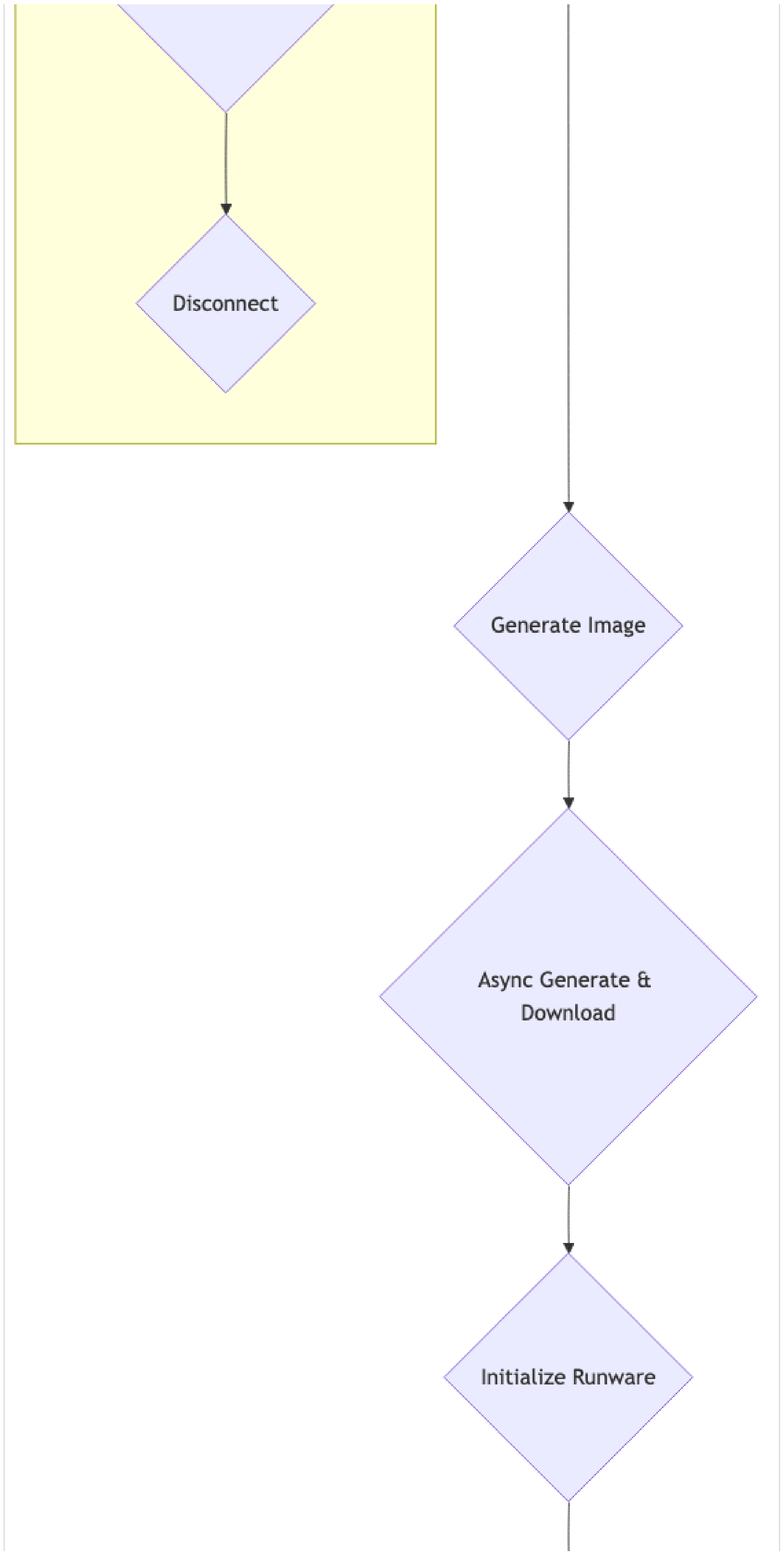
Pseudocode:

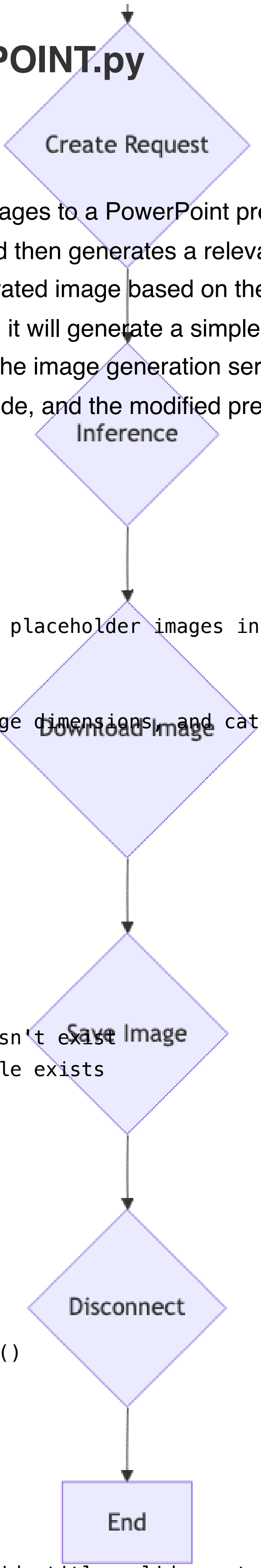
```
START
    // Runware Image Generation Module
    // This module provides functions to generate and download images using the Runware API.

    // Async Function
    FUNCTION generate_and_download_image(prompt, output_path, ...):
        INITIALIZE Runware client
        CREATE image request
        TRY
            images = runware.imageInference(requestImage)
            FOR each image:
                DOWNLOAD the image
                SAVE it to a file
                ADD the file path to a list
        FINALLY
            DISCONNECT from Runware API
        END TRY
        RETURN list of generated file paths
    END FUNCTION

    // Sync Wrapper
    FUNCTION generate_image(prompt, output_path, ...):
        RUN the async function generate_and_download_image
        RETURN the result
    END FUNCTION

    // Parallel Async Function
    FUNCTION generate_multiple_images_parallel(prompts_and_paths, ...):
        INITIALIZE Runware client
        PREPARE all image requests
        TRY
            all_results = asyncio.gather(all_requests)
            FOR each result:
                DOWNLOAD and SAVE the images
        FINALLY
```

```
                DISCONNECT from Runware API
            END TRY
            RETURN list of generated file paths
        END FUNCTION

        // Parallel Sync Wrapper
        FUNCTION generate_images_parallel(prompts_and_paths, ...):
            RUN the async function generate_multiple_images_parallel
            RETURN the result
        END FUNCTION
    END
```

Mermaid Diagram:


Parallel Generation
Generate Images Parallel
Async Parallel Generation
Prepare All Requests
Gather All Results
Download All Images
Start


Disconnect
Generate Image
Async Generate &
Download
Initialize Runware



b06_IMAGES_FOR_POWERPOINT.py

Purpose:

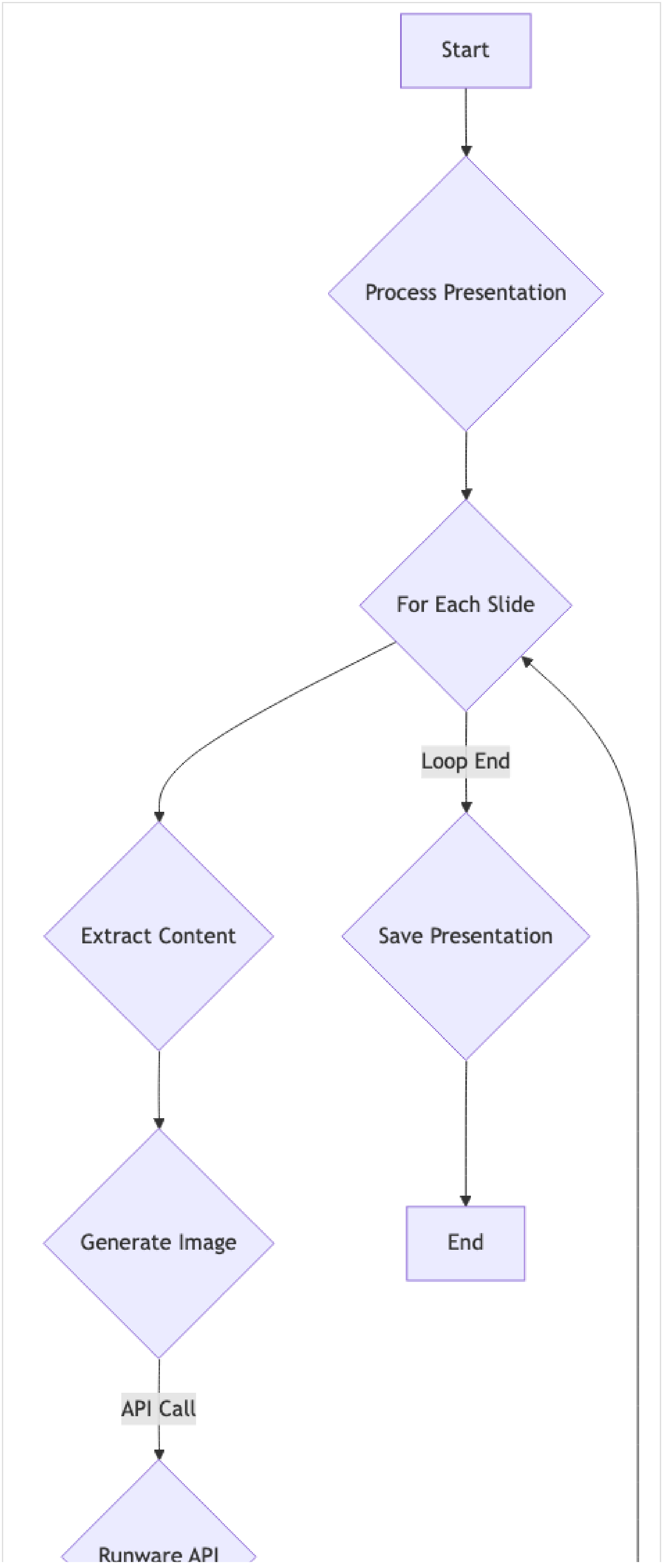
This script is designed to automatically add images to a PowerPoint presentation. It iterates through each slide of the presentation, extracts the title and content, and then generates a relevant image. The script is configured to call the Runware API to create a high-quality, AI-generated image based on the slide's content. However, it also includes a robust fallback mechanism: if the API call fails, it will generate a simple placeholder image locally. This ensures that every slide will have a visual element, even if the image generation service is unavailable. Once an image is generated or retrieved, it is inserted into the slide, and the modified presentation is saved as a new file.

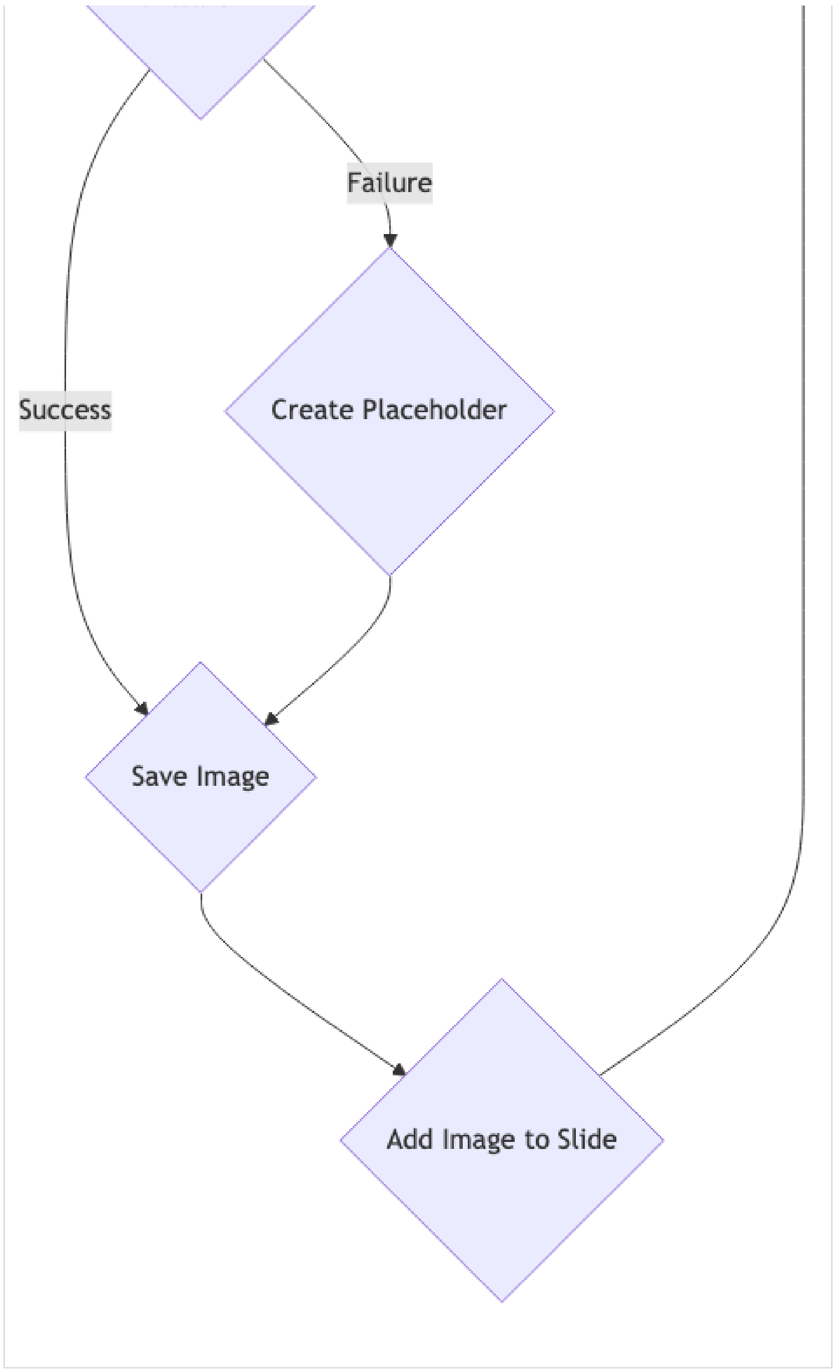
Pseudocode:

```
START
    // PowerPoint Image Generator
    // This script generates and inserts placeholder images into a PowerPoint presentation.

    // Initialization
    DEFINE constants for file paths, image dimensions, and category styles

    // Main Function
    FUNCTION main():
        setup()
        process_presentation()
    END FUNCTION

    // Helper Functions
    FUNCTION setup():
        CREATE image directory if it doesn't exist
        CHECK if the input PowerPoint file exists
    END FUNCTION

    FUNCTION process_presentation():
        LOAD the presentation
        FOR each slide (up to a limit):
            EXTRACT slide content
            DEFINE image file path
            IF image doesn't exist THEN
                generate_image_for_slide()
            END IF
            add_image_to_slide()
        END FOR
        SAVE the modified presentation
    END FUNCTION

    FUNCTION generate_image_for_slide(slide_title, slide_content, output_path):
        CREATE a prompt from the slide content
        CALL the Runware API to generate the image
        IF API call fails THEN
```

```
            CREATE a local placeholder image
        END IF
    END FUNCTION

    FUNCTION create_placeholder_image(title, category, width, height):
        CREATE a new image with category-specific styling
        DRAW an icon and the title on the image
        RETURN the image
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
Process Presentation
For Each Slide
Loop End
Extract Content
Save Presentation
Generate Image
End
API Call
Runware API



Utility and Testing Scripts

a03_TEST_LLM.py

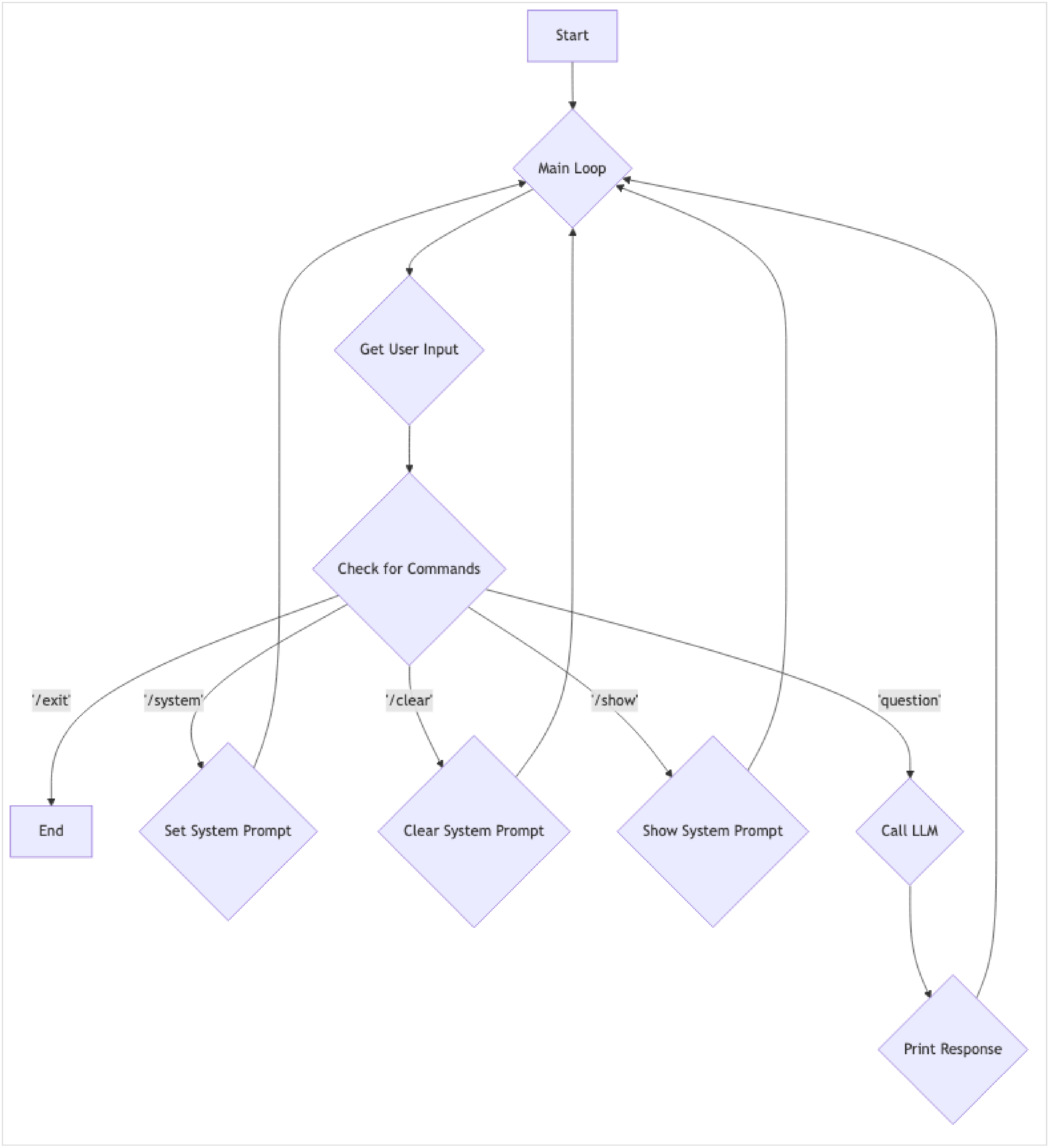
Purpose:

This script provides an interactive command-line interface for testing and interacting with the Large Language Model (LLM) accessed through the `a02_LLM_Access.py` module. It allows users to enter questions directly, and it also supports special commands for managing a persistent system prompt. Users can set, clear, and view the system prompt, which is useful for guiding the LLM's behavior across multiple interactions. The script continuously prompts for input, sends the user's questions to the LLM, and displays the returned response, making it a convenient tool for experimenting with different prompts and system messages.

Pseudocode:

```
START
    // Test LLM Interaction Script
    // This script provides an interactive command-line interface to test the LLM.

    // Initialization
    IMPORT call_llm function from a02_LLM_Access module

    // Main Function
    FUNCTION main():
        PRINT "LLM Question-Answering System" banner
        INITIALIZE system_prompt to None

        LOOP indefinitely:
            GET user_input
            IF user_input is an exit command THEN
                BREAK loop
            ELSE IF user_input is a system prompt command THEN
                SET system_prompt
            ELSE IF user_input is a clear command THEN
                CLEAR system_prompt
            ELSE IF user_input is a show command THEN
                SHOW system_prompt
            ELSE
                SEND user_input to call_llm with system_prompt
                PRINT the response
            END IF
        END LOOP
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:



atest_runware_integration.py

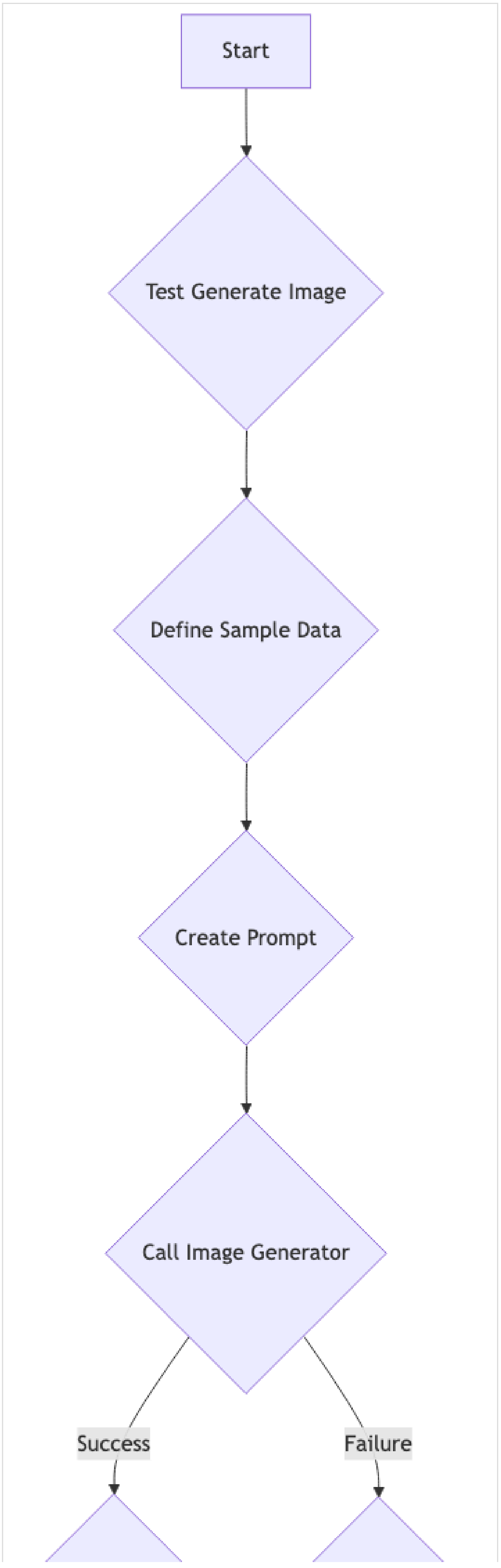
Purpose:

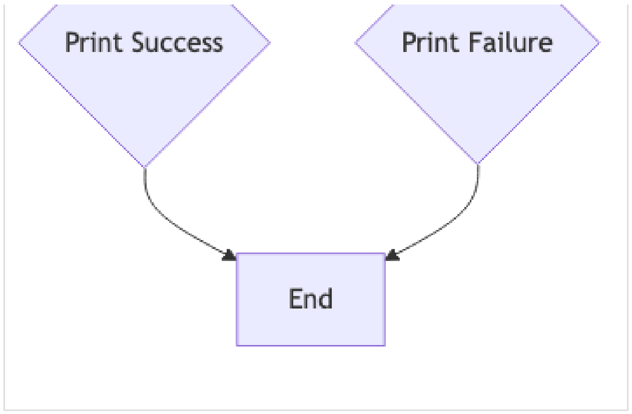
This script serves as a simple integration test for the `runware_image_generator.py` module. Its purpose is to demonstrate how to use the image generation function to create an image for a hypothetical PowerPoint slide. It defines a sample slide title and content, constructs a descriptive prompt from this information, and then calls the `generate_image` function to create and download the image. The script provides a clear example of how the image generation module can be used in a practical application, and it serves as a quick way to verify that the Runware API integration is working correctly.

Pseudocode:

```
START
    // Test script for Runware image generator
    // This script demonstrates how to use the image generator for a PowerPoint slide.

    // Test Function
    FUNCTION test_generate_slide_image():
        DEFINE sample slide title and content
        CREATE an output directory
        CREATE a prompt from the slide content
        PRINT the prompt

        CALL generate_image from the runware_image_generator module
        IF images are downloaded THEN
            PRINT success message
        ELSE
            PRINT failure message
        END IF
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        test_generate_slide_image()
    END IF
END
```

Mermaid Diagram:


Start
Test Generate Image
Define Sample Data
Create Prompt
Call Image Generator
Success
Failure



rename_files.py

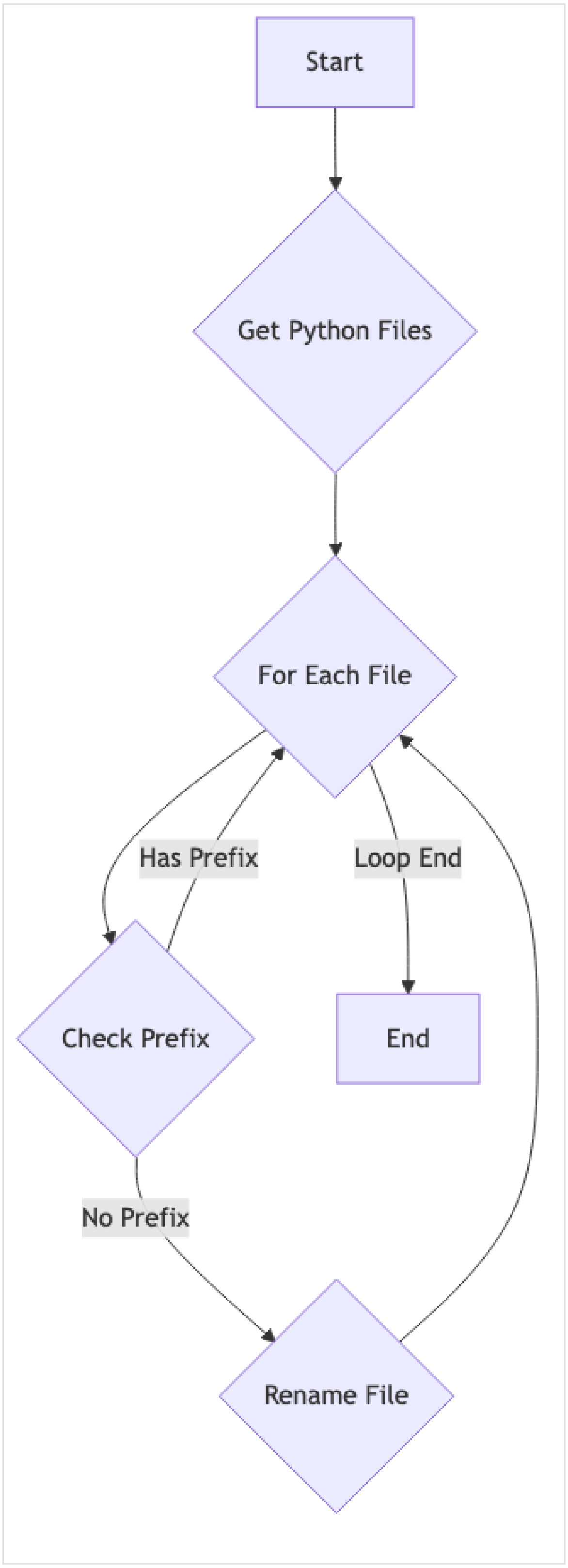
Purpose:

This is a utility script for renaming files within the project. Its purpose is to prepend a specified prefix to the name of every Python file in the current directory. The script is designed to be run directly from the command line. It first identifies all `.py` files, then iterates through them, adding the prefix to each one that doesn't already have it. This is useful for enforcing a consistent naming convention across the project or for organizing files into a specific order.

Pseudocode:

```
START
    // Rename Files Script
    // This script renames all Python files in a directory to have a specified prefix.

    // Main Function
    FUNCTION rename_files_with_prefix(directory, prefix):
        GET list of Python files in the directory
        REMOVE this script from the list
        CONFIRM with the user before renaming

        FOR each file in the list:
            IF the file does not already have the prefix THEN
                RENAME the file
            END IF
        END FOR
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        rename_files_with_prefix(current_directory, "a")
    END IF
END
```

Mermaid Diagram:



test_01_RAG.py

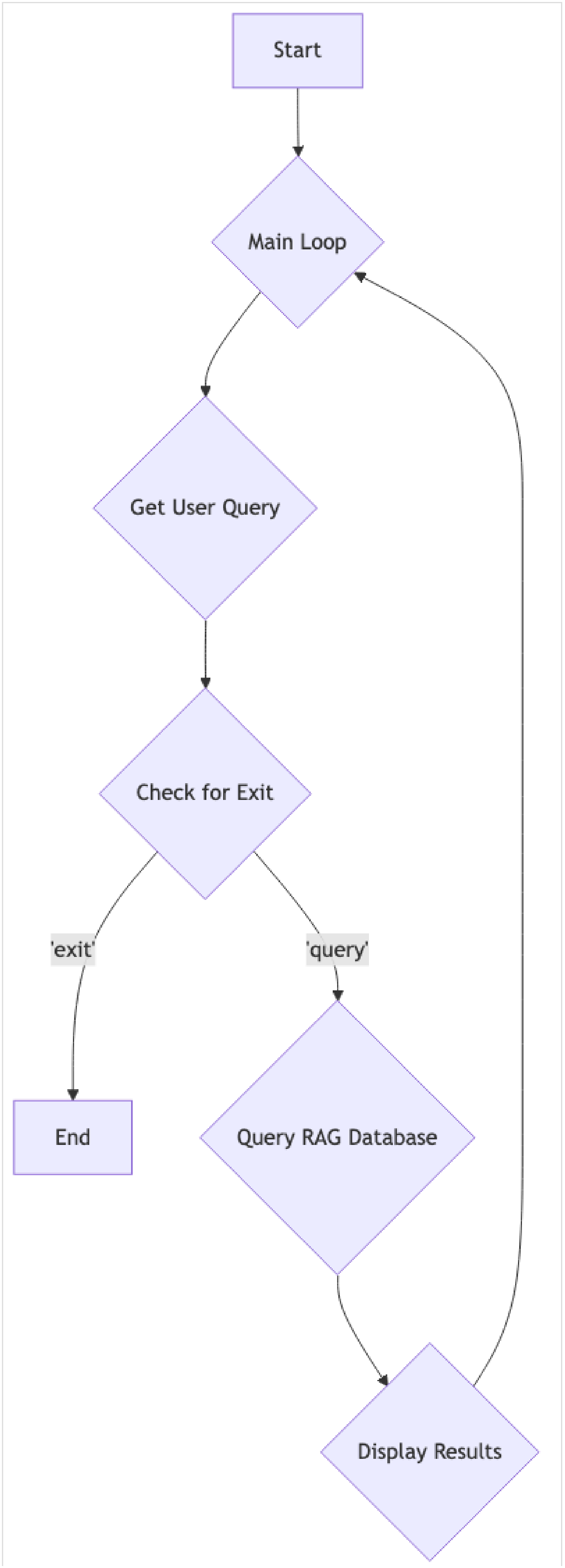
Purpose:

This script provides a user-friendly, interactive command-line interface for querying the RAG (Retrieval Augmented Generation) database that was created by the `a01_RAG_DB_Creation_PDF.py` script. It allows users to ask questions in natural language about the content of the PDF documents that were processed into the database. The script takes the user's query, sends it to the ChromaDB vector database, and retrieves the most relevant text chunks. It then displays these results to the user, along with metadata such as the document ID and a relevance score, making it easy to find information within the source documents.

Pseudocode:

```
START
    // RAG Database Query Interface
    // This script provides an interactive interface to query the RAG database.

    // Main Function
    FUNCTION main():
        PRINT "Cisco AI PDF RAG Database Query Interface" banner
        LOOP indefinitely:
            GET user query
            IF query is an exit command THEN
                BREAK loop
            END IF

            results = query_rag_database(query)
            IF results are returned THEN
                display_results(results, query)
            END IF
        END LOOP
    END FUNCTION

    // Helper Functions
    FUNCTION query_rag_database(query_text, num_results):
        INITIALIZE ChromaDB client
        GET the collection
        QUERY the collection with the user's text
        RETURN the results
    END FUNCTION

    FUNCTION display_results(results, query_text):
        PRINT the query
        FOR each result:
            PRINT the document ID and relevance score
            PRINT a preview of the document content
        END FOR
    END FUNCTION

    // Script Execution
    IF script is run directly THEN
        main()
    END IF
END
```

Mermaid Diagram:


Start
Main Loop
Get User Query
Check for Exit
'exit'
'query'
End
Query RAG Database
Display Results

Placeholder Files

a09_Flash_Card.py

Purpose:

This file is a placeholder for a future script that will generate flash cards from the course content.

a11_Video_Generation_for_slides.py

Purpose:

This file is a placeholder for a future script that will generate a video from the presentation slides.