Project Overview

# 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

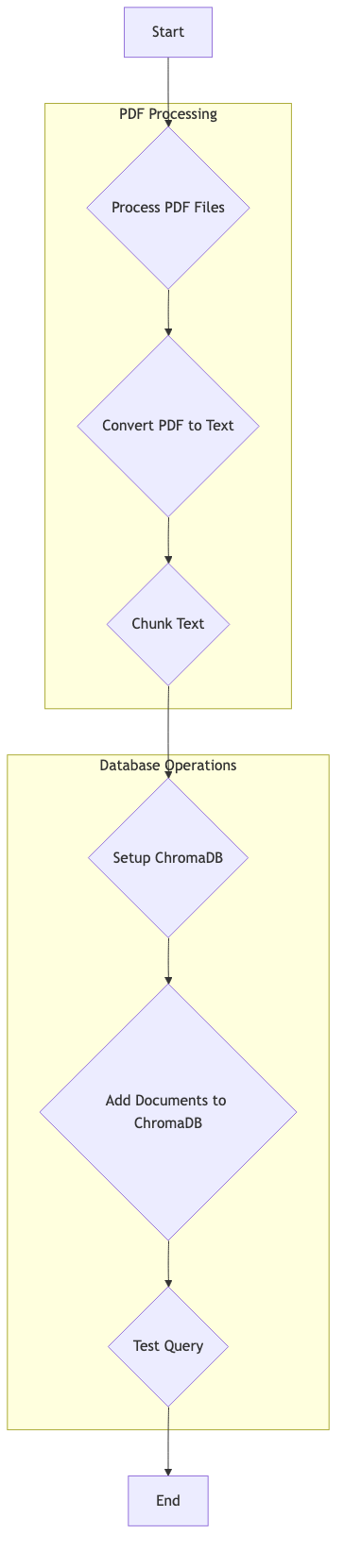
### Purpose:

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:



## a02\_LLM\_Access.py

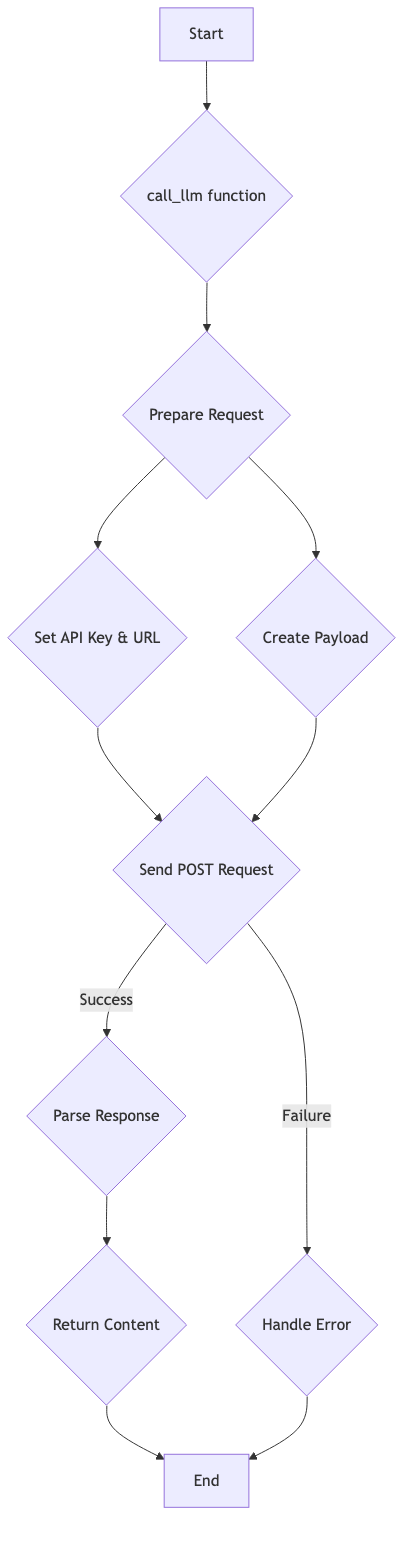
### Purpose:

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:



## a04\_CREATE\_OUTLINE.py

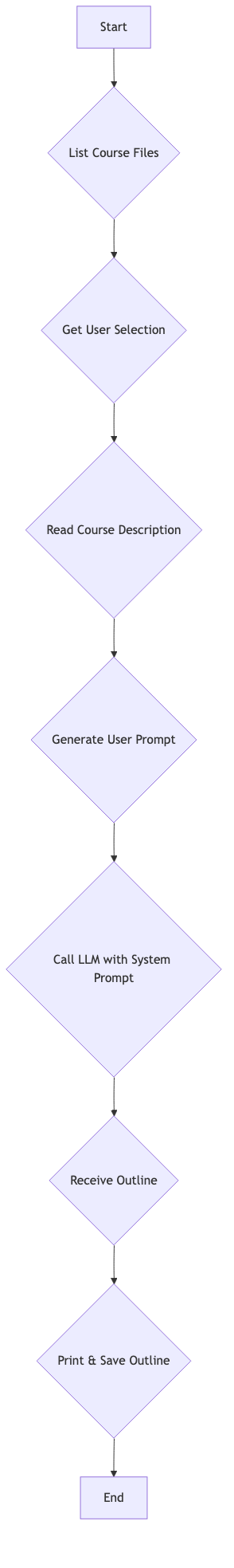
### Purpose:

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:



## a06-Student\_Notes\_Student\_Handbook.py

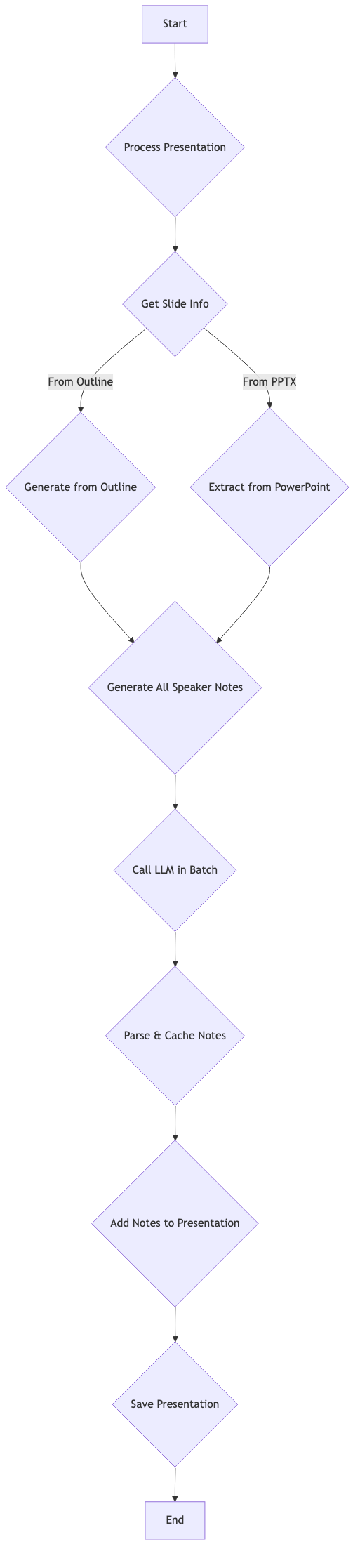
### Purpose:

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:



## a07\_QUIZ\_Per\_Module.py

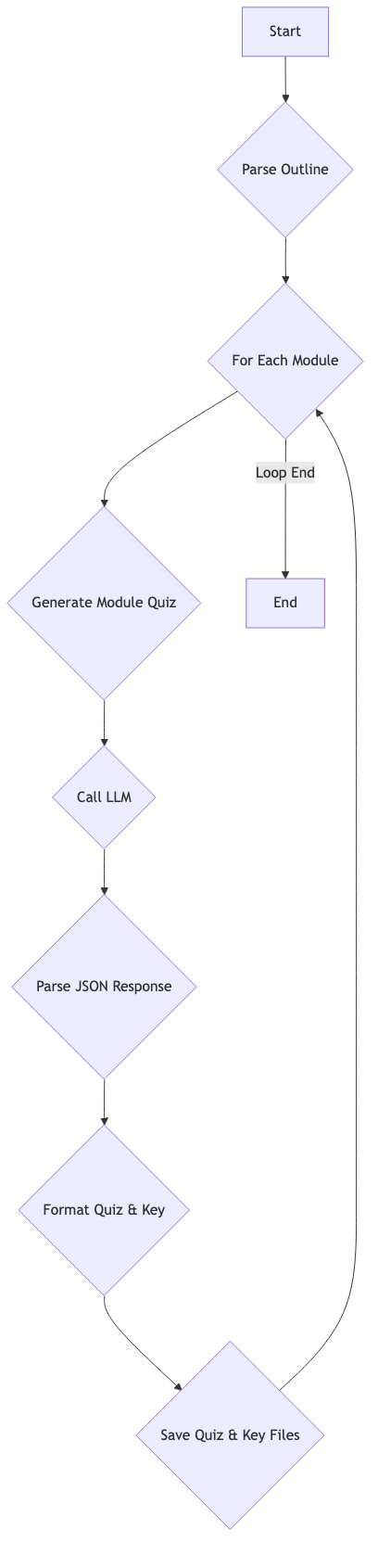
### Purpose:

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:



## a08\_Final\_Exam.py

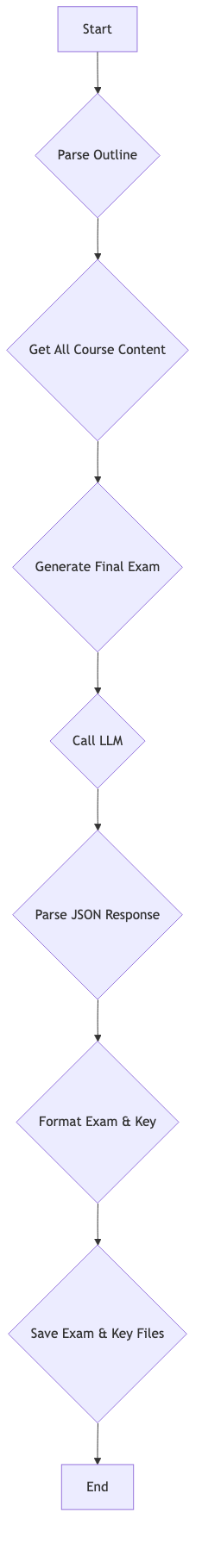
### Purpose:

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:



# PowerPoint & Asset Creation

## a05\_CREATE\_POWERPOINT.py

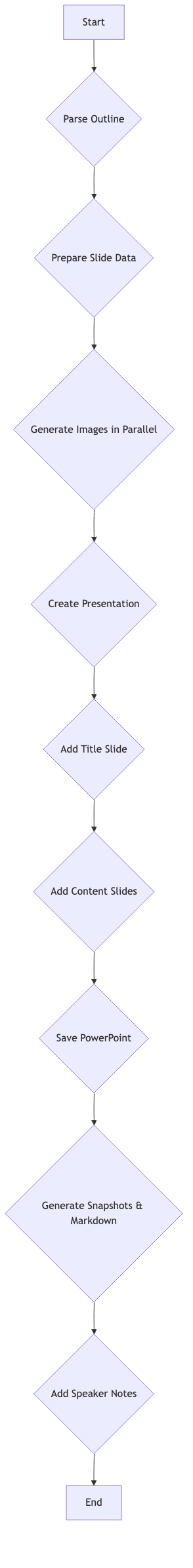
### Purpose:

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:



## a06\_Image\_Generation.py

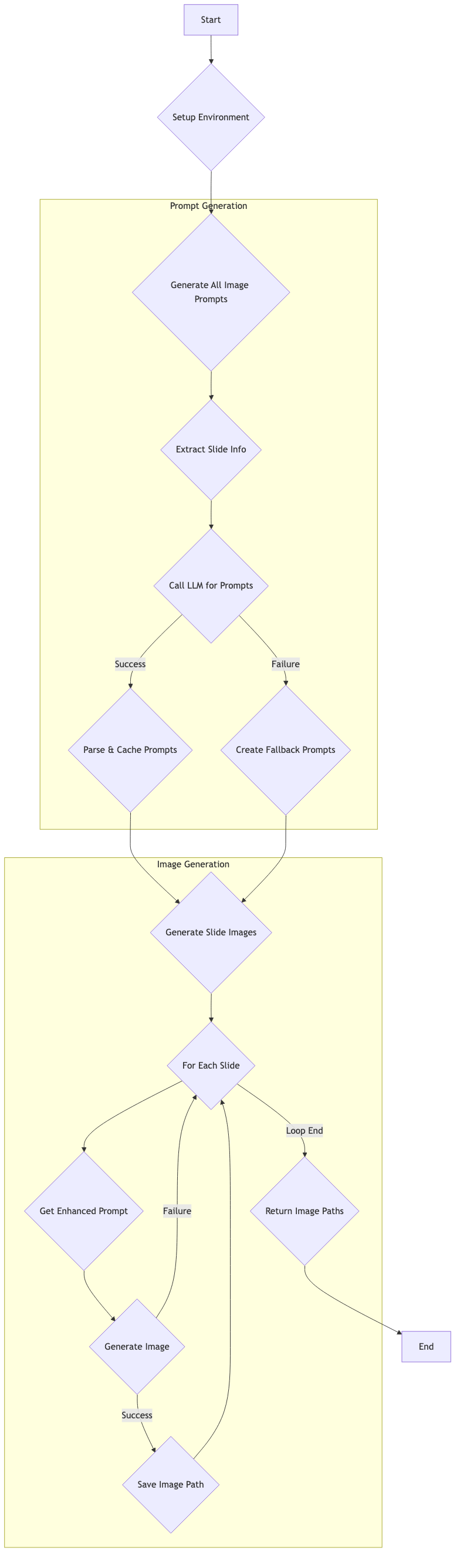
### Purpose:

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:



## a07\_Slide\_Snapshot\_Generator.py

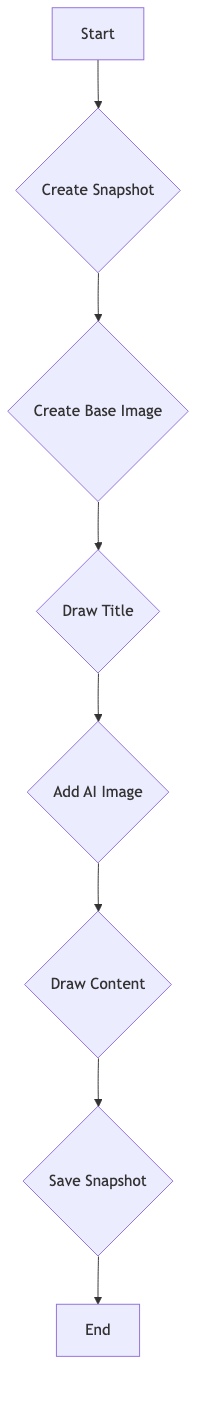
### Purpose:

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:



## a10\_Audio\_Generation\_for\_Slides.py

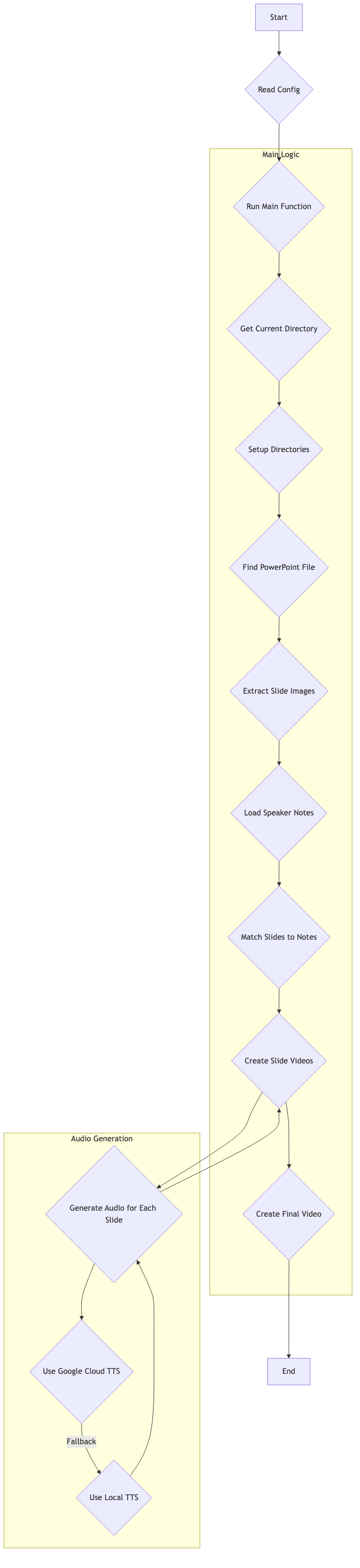
### Purpose:

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:



## arunware\_image\_generator.py

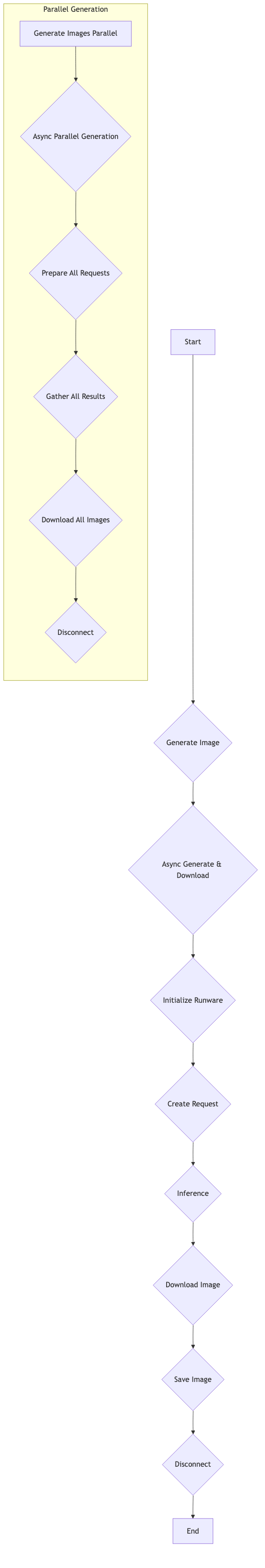
### Purpose:

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:



## b06\_IMAGES\_FOR\_POWERPOINT.py

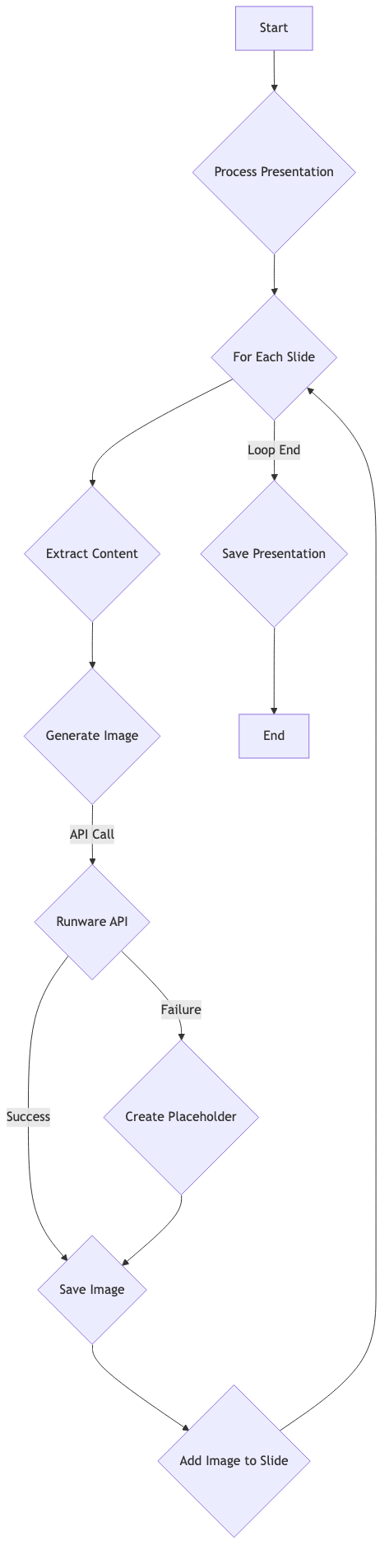
### Purpose:

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:



# Utility and Testing Scripts

## a03\_TEST\_LLM.py

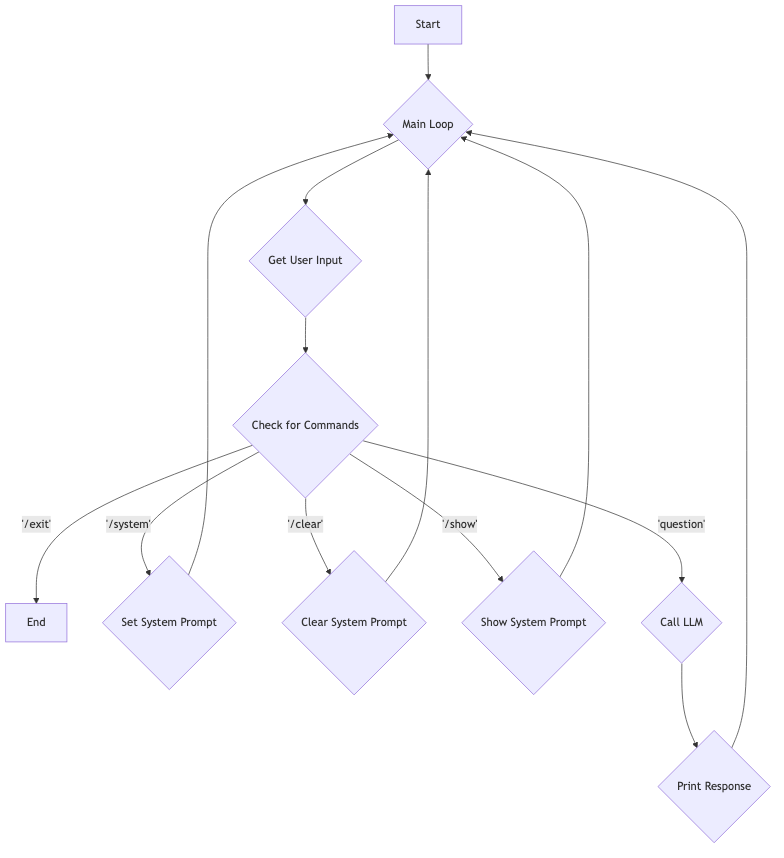
### Purpose:

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

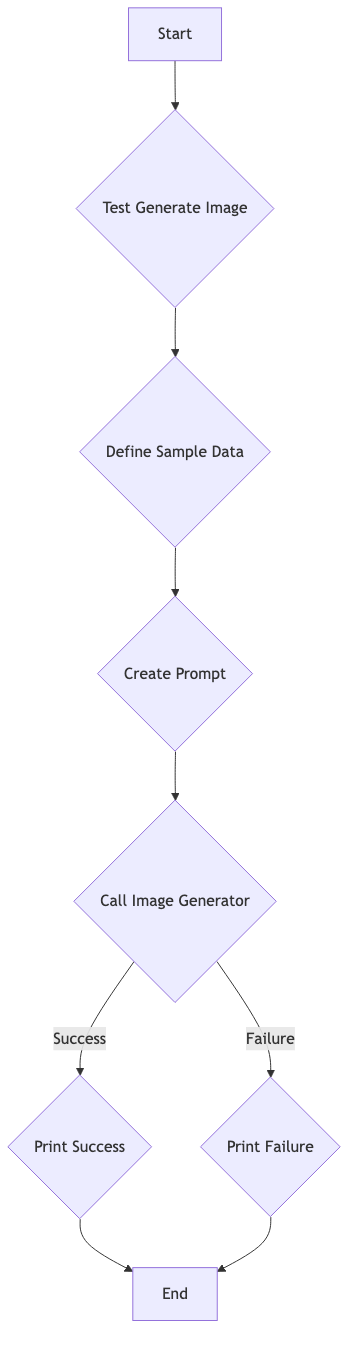
### Purpose:

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:



## rename\_files.py

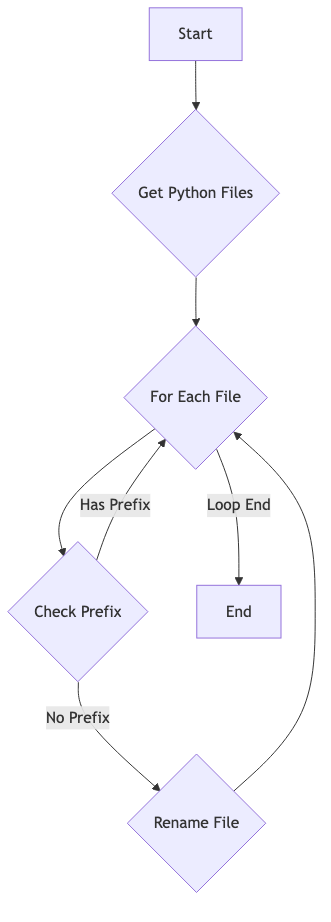
### Purpose:

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

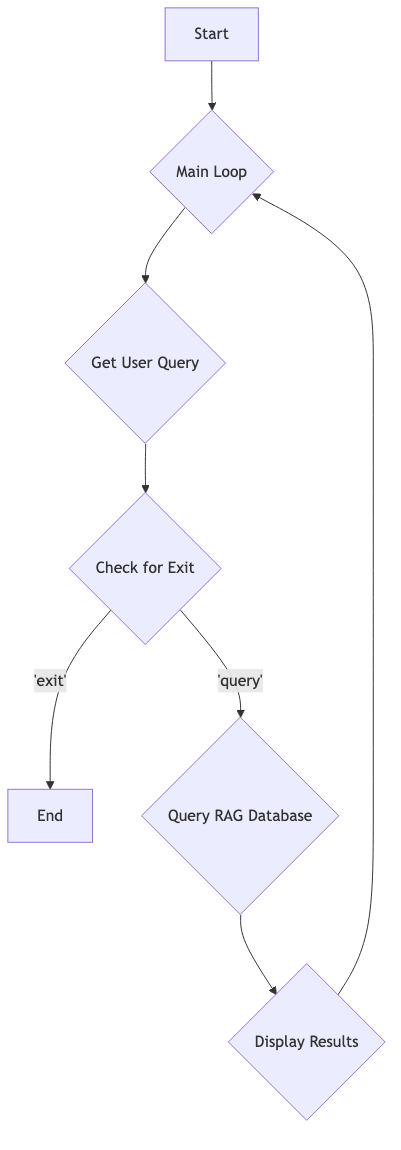
### Purpose:

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:



# 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.