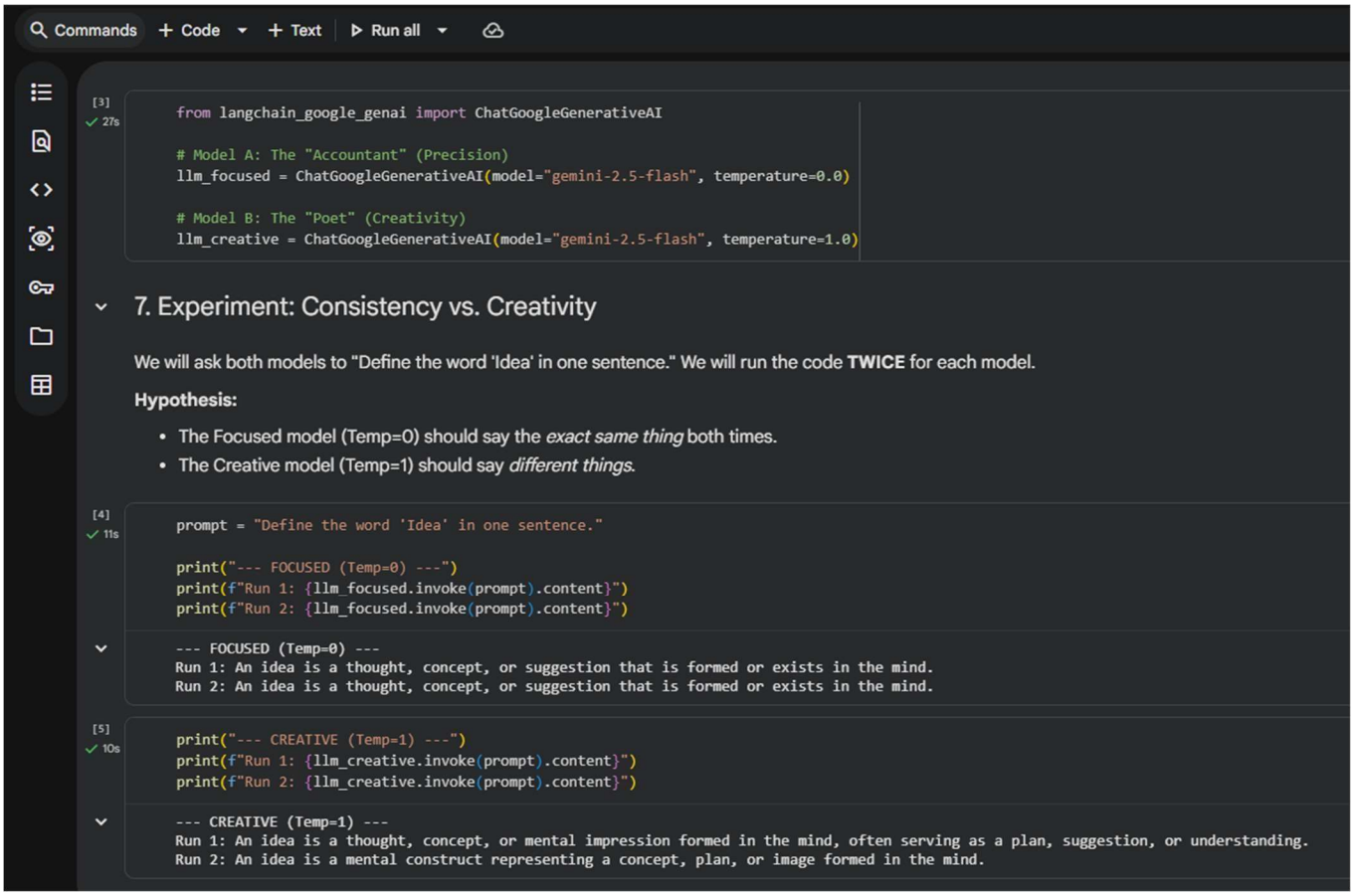
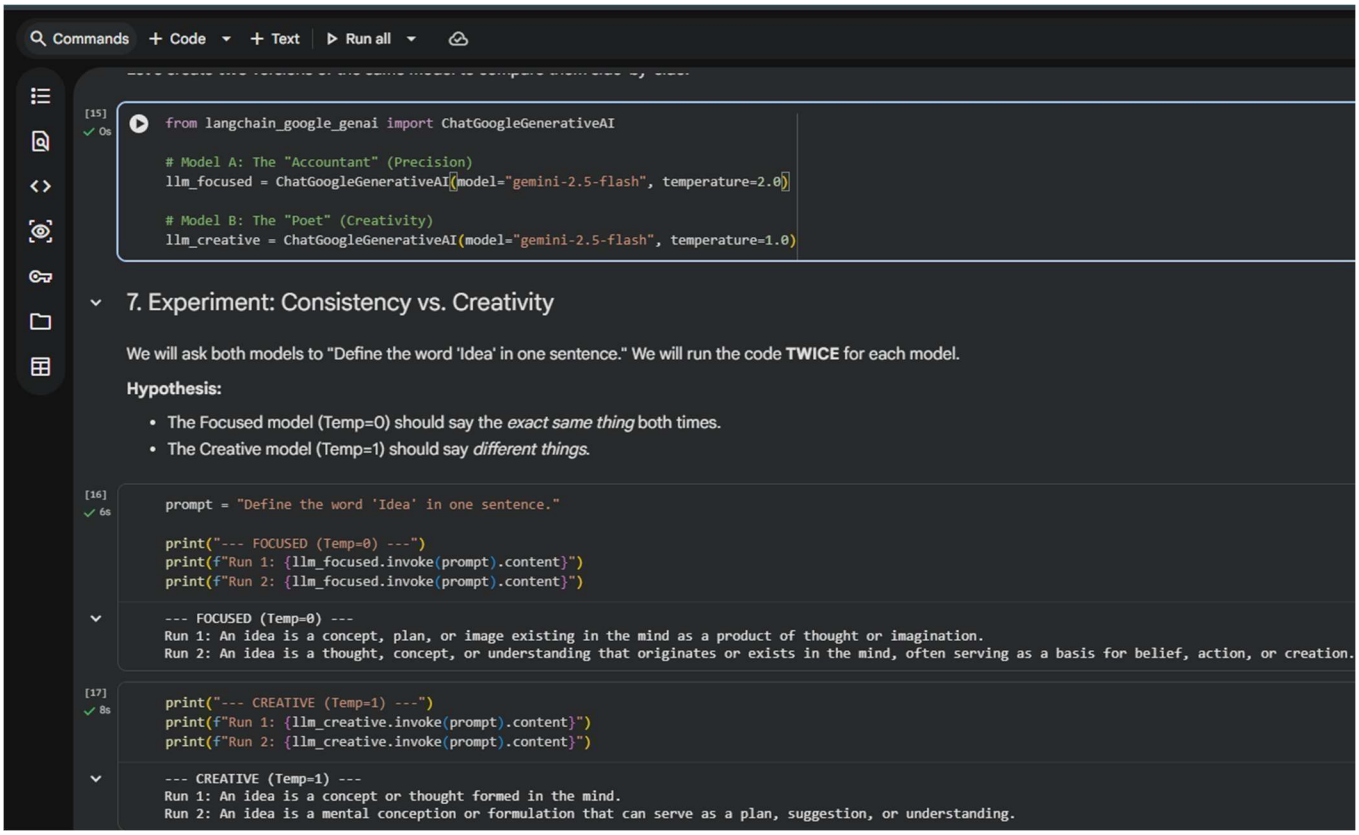
GENERATIVE AI AND ITS APPLICATIONS - UE23CS342BA4

Hands-on 2

NAME : GREESHMA YP
SRN : PES2UG22CS207
SEM : 6
SEC : A
DATE : 23-01-2026

Notebook: Lang Chain Founda on





Main Difference :

In the first image, the focused model is set with temperature = 0.0, which makes it produce identical outputs every time (stable and consistent behavior).
In the second image, the focused model uses temperature = 2.0, which causes the outputs to vary each time (more creative and random responses).
So the only significant change is the temperature value, and that alone shifts the model from predictable behavior to creative behavior.

Setup Cell
What it does:
Installs necessary libraries (LangChain, Google GenAI, dotenv, etc.)
Loads environment variables
Prompts for Google API key
Initializes the Gemini LLM object

This cell sets up the entire environment required to use Gemini through LangChain. It basically establishes the connection between your notebook and the AI model.

Basic LLM Call What
it does:
Sends a simple prompt to the model
Receives and displays the output
This is just a basic test to confirm that the model is functioning correctly.

Using ChatPromptTemplate What it
does:
Builds a structured prompt template
Accepts dynamic inputs (such as name or topic)
Sends the formatted prompt to the model
Instead of writing a new prompt each time, this allows you to reuse a template and insert different values dynamically.

LCEL (LangChain Expression Language) Chain What it
does:
Connects Prompt → Model → Output Parser using the | operator
Forms a processing pipeline Example:
Prompt | LLM | OutputParser
This creates a smooth workflow where the output from one step automatically becomes the input for the next step.

Temperature Experiment (Consistency vs Creativity) What it
does:
Creates two models:
Temperature = 0 (more focused)
Temperature = 1 (more creative)
Executes the same prompt twice
Compares the results
This demonstrates how temperature controls randomness:
Lower temperature → more consistent output
Higher temperature → more variation in output

Assignment (Movie Example – One Line LCEL) What it
does:
Accepts a movie name
Asks for its release year
Calculates how many years ago it was
Uses a single LCEL pipeline
This example shows how reasoning and chaining can be combined into one clean workflow.

Notebook 2: Prompt Engineering

Cell 1 – Installation / Setup What it
does:
Installs required packages
Imports necessary modules
Loads environment variables
Requests Google API key
Initializes the LLM
This prepares the environment for experimenting with different prompting techniques.

Cell 2 – Basic Prompt Example
What it does:
Sends a simple instruction to the model
Prints the response
This verifies that the model is responding correctly to basic prompts.

Cell 3 – Zero-Shot Prompting What
it does:
Provides instructions without examples
Directly asks the model to perform a task
This tests the model's ability to complete a task based only on instructions.

Cell 4 – Few-Shot Prompting (Manual) What it
does:
Includes a few example input-output pairs
Adds a new input afterward

Model follows the demonstrated pattern
This method guides the model by showing examples before asking it to solve a similar task.

Cell 5 – FewShotChatMessagePromptTemplate What it
does:
Implements structured few-shot prompting Separates:
System instructions
Example conversations (Human → AI)
New query
Uses LangChain's template system
Rather than placing everything in one long text, this organizes prompts clearly using message roles, making it easier for the model to interpret.

Cell 6 – Executing the Few-Shot Chain What it
does:
Builds an LCEL chain:
Prompt | LLM | OutputParser
Runs the chain with a new input
This executes the structured few-shot prompt and displays the result.

Cell 7 – Analysis Section What
it does:
Explains the advantages of structured few-shot prompting
Mentions the attention mechanism
Discusses clarity and reliability
This part highlights how structured prompting improves focus and response quality.

Notebook 3: Advanced Prompting Techniques

This notebook compares advanced reasoning methods:
Simple Prompting
Chain of Thought (CoT)
Tree of Thought (ToT)
Graph of Thought (GoT)

Cell 1 – Installation & Setup What
it does:
Installs required packages
Imports necessary modules
Loads environment variables
Configures the Google API key
Creates the Gemini model instance
This prepares the notebook for testing advanced prompting strategies.

Cell 2 – Simple Prompting What it
does:
Asks a direct question
Receives an immediate answer
This represents basic prompting without detailed reasoning.

Cell 3 – Chain of Thought (CoT)
What it does:
Adds instructions like “Think step by step”
Encourages detailed reasoning
This method makes the model show intermediate reasoning steps before giving the final answer.

Cell 4 – Tree of Thought (ToT)
What it does:
Encourages exploring multiple reasoning paths
Evaluates different options before selecting the best one
Instead of following a single path, the model considers several approaches before deciding.

Cell 5 – Graph of Thought (GoT)
What it does:
Uses interconnected reasoning
Creates complex logical relationships
Allows flexible navigation between ideas
This method represents reasoning as a network rather than a straight sequence.

Cell 6 – Comparison Table
What it does: Compares:

Structure
Best use cases
Cost and latency
Provides a recommendation
This section summarizes when each reasoning strategy should be used.

Notebook 4: RAG and Vector Stores

This notebook builds a Retrieval-Augmented Generation (RAG) system using embeddings and a vector database.

Cell 1 – Package Installation What
it does:
Installs: langchain
langchain-google-genai
langchain-huggingface
sentence-transformers
faiss-cpu python-
dotenv
This prepares tools for embeddings, storage, and AI integration.

Cell 2 – Import & Setup What it
does:
Imports required modules
Loads environment variables
Requests Google API key
Initializes:
Gemini LLM
HuggingFace embedding model
This connects the notebook to both the language model and the embedding model.

Cell 3 – Create Sample Documents What it
does:
Defines text documents
Converts them into LangChain Document objects This
forms the knowledge base for the RAG system.

Cell 4 – Generate Embeddings
What it does:
Converts text into vector representations using HuggingFace embeddings This enables mathematical comparison of semantic similarity.

Cell 5 – Create FAISS Vector Store What it does:
Stores document vectors in FAISS
Builds a searchable vector database
This allows fast similarity-based document retrieval.

Cell 6 – Similarity Search What it does:
Accepts a query
Retrieves the most relevant documents
This fetches context related to the user's question.

Cell 7 – Build Retriever + LLM Chain (RAG)
What it does: Combines:
Retriever
Prompt template
LLM
Passes retrieved context to the model
Instead of answering blindly, the model now uses retrieved information to generate responses.

Cell 8 – Final Query Execution
What it does:
Takes a user question
Retrieves relevant documents
Produces a context-based answer
This demonstrates the complete RAG workflow:
User Query → Retrieve Relevant Documents → Generate Answer