DV Analytics
Transforming You

LangChain Part1 – Basic Chains

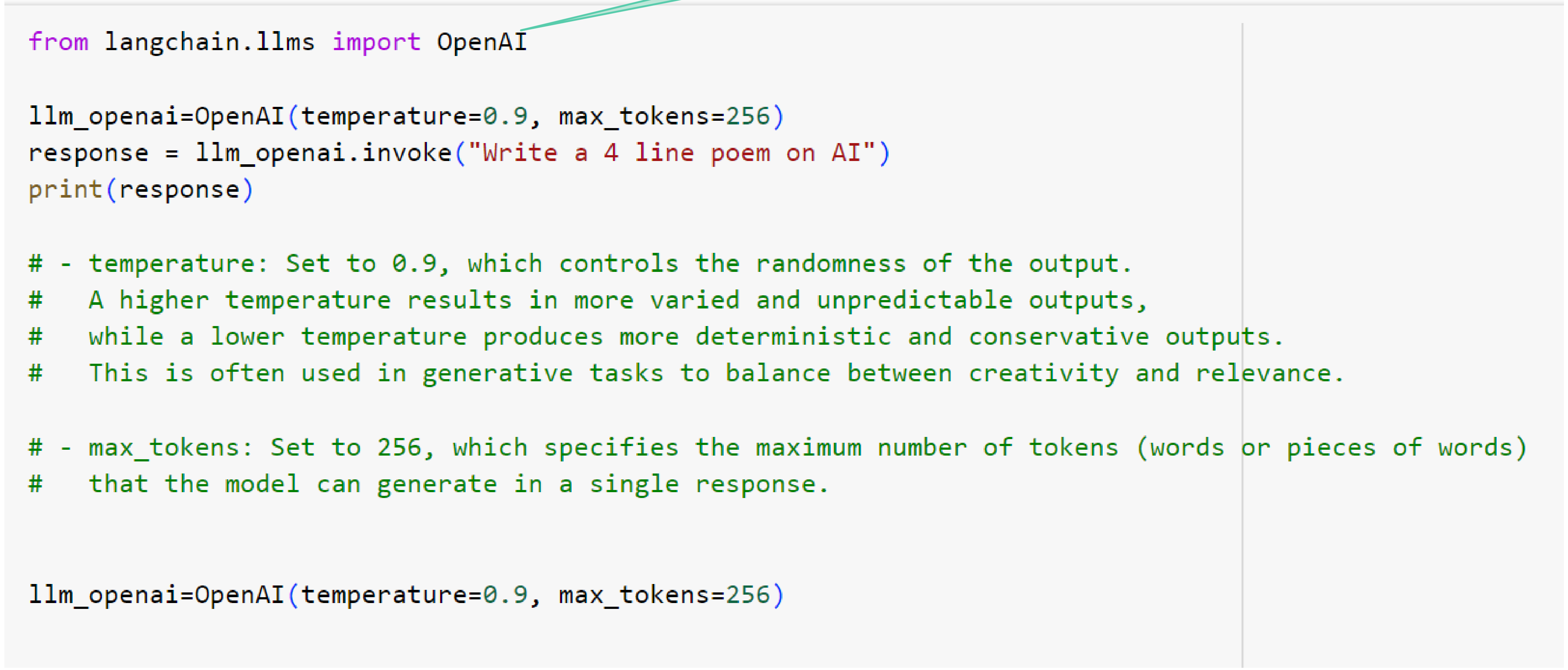
Venkata Reddy AI Classes
https://www.youtube.com/@VenkataReddyAIClasses/playlists

Contents

- Where LLMs Fail?
- LangChain is an LLM Framework.
- LangChain Components
- Prompt Templates
- Sequential Chain
- IDP (Intelligent Document Processing)
- Book Summary App

LLMs Recap

Open AI is a paid service.



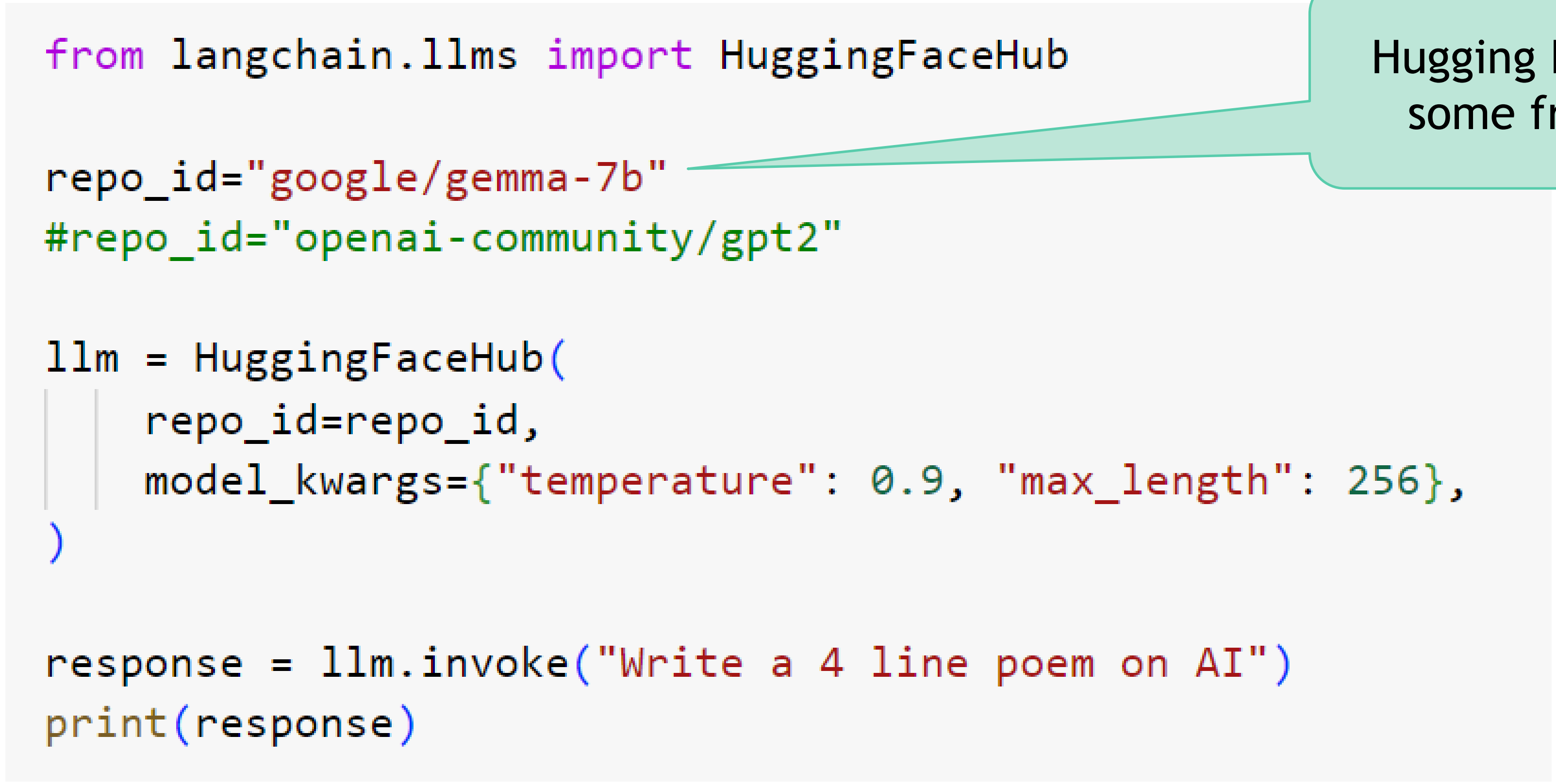
```python
from langchain.llms import OpenAI

llm_openai=OpenAI(temperature=0.9, max_tokens=256)
response = llm_openai.invoke("Write a 4 line poem on AI")
print(response)

# - temperature: Set to 0.9, which controls the randomness of the output.
#   A higher temperature results in more varied and unpredictable outputs,
#   while a lower temperature produces more deterministic and conservative outputs.
#   This is often used in generative tasks to balance between creativity and relevance.

# - max_tokens: Set to 256, which specifies the maximum number of tokens (words or pieces of words)
#   that the model can generate in a single response.


llm_openai=OpenAI(temperature=0.9, max_tokens=256)
```

Cohere - Best Open source Alternative to OpenAI

```python
from langchain.llms import Cohere

llm = Cohere(model="command-xlarge-nightly")
response = llm.invoke("Write a 4 line poem on AI")
print(response)
```

```
AI is a wondrous force,
In code, it's a wondrous course,
Though machines may fail,
We must still learn and adore!
```

LLMs from Hugging Face



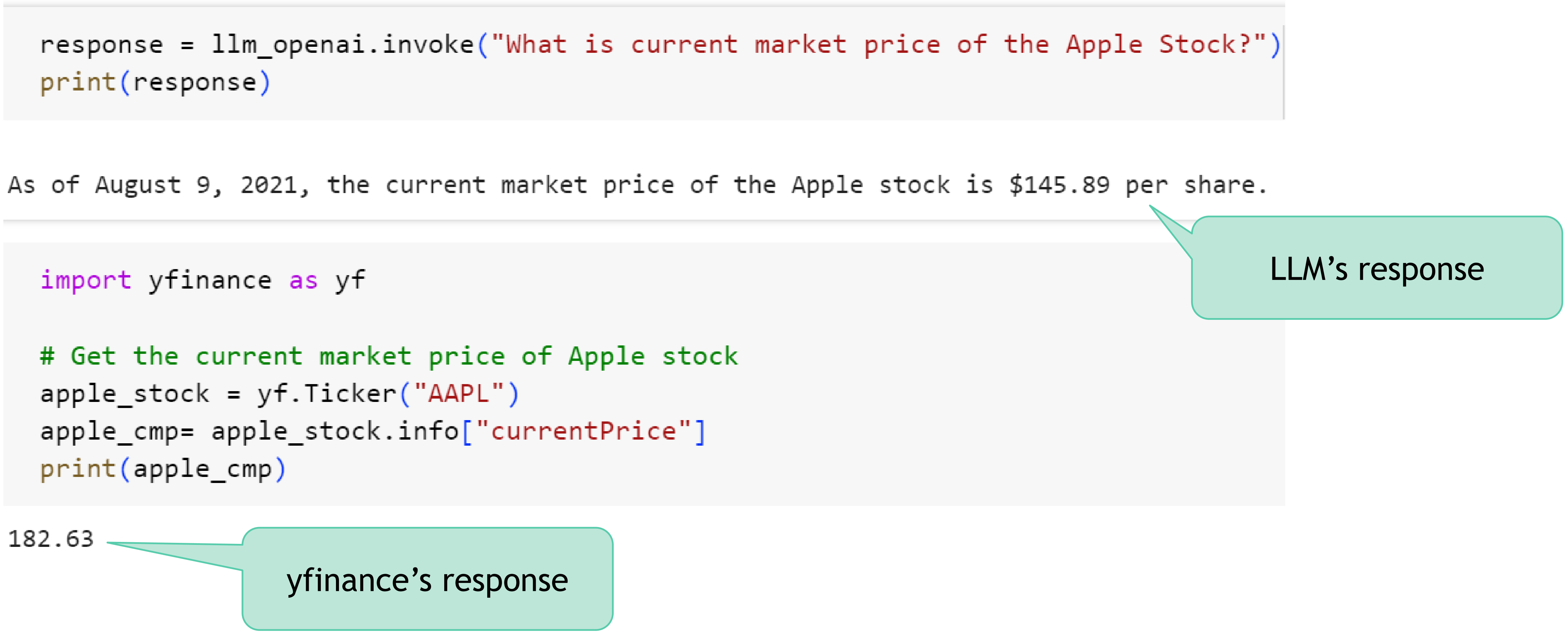
```python
from langchain.llms import HuggingFaceHub

repo_id="google/gemma-7b"
#repo_id="openai-community/gpt2"

llm = HuggingFaceHub(
    repo_id=repo_id,
    model_kwargs={"temperature": 0.9, "max_length": 256},
)

response = llm.invoke("Write a 4 line poem on AI")
print(response)
```

Hugging Face offers some free LLMs.

Where LLMs Fail?

- **Accessing Up-to-Date Information**
  - Check news websites for today's global headlines
  - Financial apps provide current stock market data
- **Extracting Data from Specific and Private Documents**
  - Search personal email for recent flight confirmation.
  - Patent filings details aren't accessible by LLMs
- **Understanding Contextual Nuances**
  - Interpret sarcasm in text messages differently.
  - Cultural references might not be accurately interpreted
- **Handling Highly Technical or Niche Topics**
  - Specific programming bugs solutions often require updates.
  - Advanced mathematical theories could be overly simplified

Where LLMs Fail?



```
response = llm_openai.invoke("What is current market price of the Apple Stock?")
print(response)
```

```
As of August 9, 2021, the current market price of the Apple stock is $145.89 per share.
```

LLM’s response

```
import yfinance as yf

# Get the current market price of Apple stock
apple_stock = yf.Ticker("AAPL")
apple_cmp= apple_stock.info["currentPrice"]
print(apple_cmp)
```

```
182.63
```

yfinance’s response

Connecting LLMs with External world

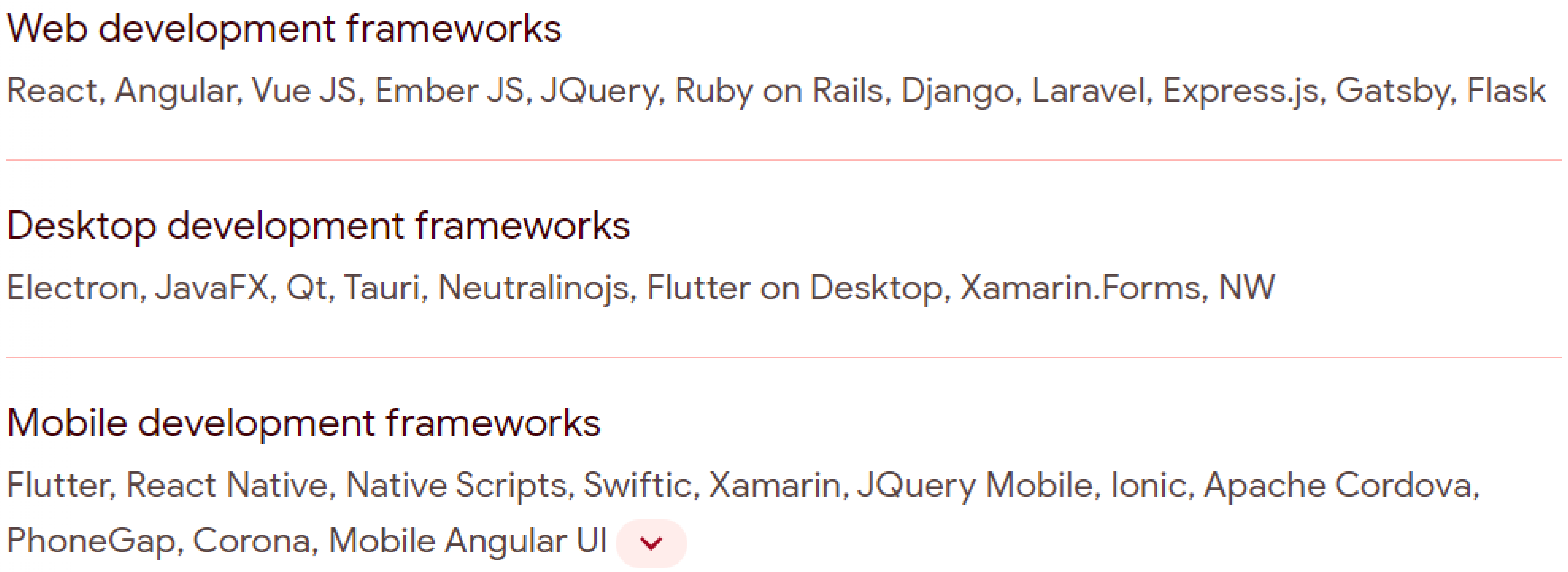
- Can we add our own information when using LLMs?
- Is it possible to update these LLMs with the latest information or data from outside?
- How can we create customized applications using LLMs?
- Can we combine these LLMs with other online services?

- Is there an easy way to work with LLM and create an app on top of it ?
  – Yes ; Using LangChain framework

LangChain is an LLM Framework.

- LangChain is an LLM Framework.
- What exactly is a Framework?
  - In software development, a "framework" is essentially a toolbox or a platform that provides a structured way for developers to build and manage their software projects.
  - Imagine you're constructing a model car; instead of creating every single piece from scratch, you start with a kit that has parts like the body, wheels, and axles. You then assemble and customize it to create your unique model car.
  - Framework includes libraries of pre-written code, software tools, and guidelines that help in the rapid development of applications

Example Frameworks



Web development frameworks

React, Angular, Vue JS, Ember JS, JQuery, Ruby on Rails, Django, Laravel, Express.js, Gatsby, Flask

Desktop development frameworks

Electron, JavaFX, Qt, Tauri, Neutralinojs, Flutter on Desktop, Xamarin.Forms, NW

Mobile development frameworks

Flutter, React Native, Native Scripts, Swiftic, Xamarin, JQuery Mobile, Ionic, Apache Cordova, PhoneGap, Corona, Mobile Angular UI

LangChain

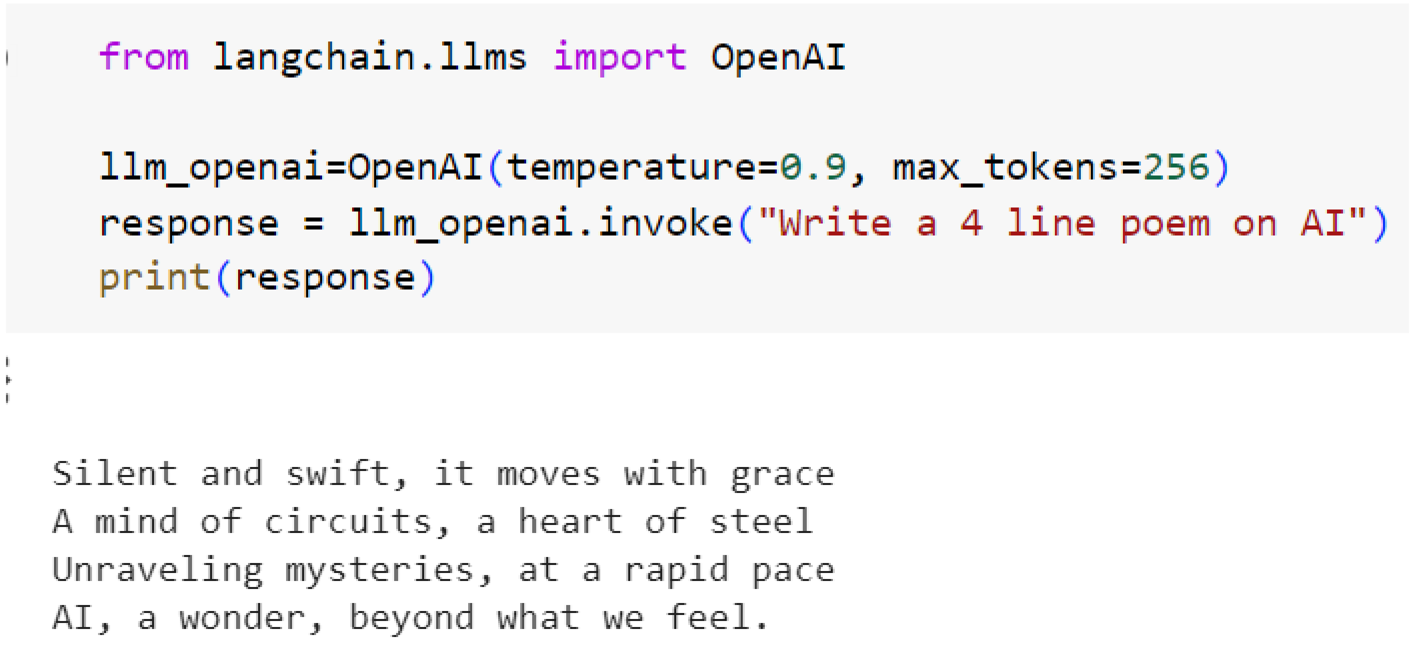
- LangChain is a framework for building applications powered by large language models (LLMs).
- It facilitates combining LLMs with various data sources and functionalities
- It gives you much more power than using a simple LLM API from openai or huggingface

Basic LangChain Components

1. A prompt Template
2. An LLM model
3. A Chain

We will get into other components like memory, retrieval, agents, tools etc., later

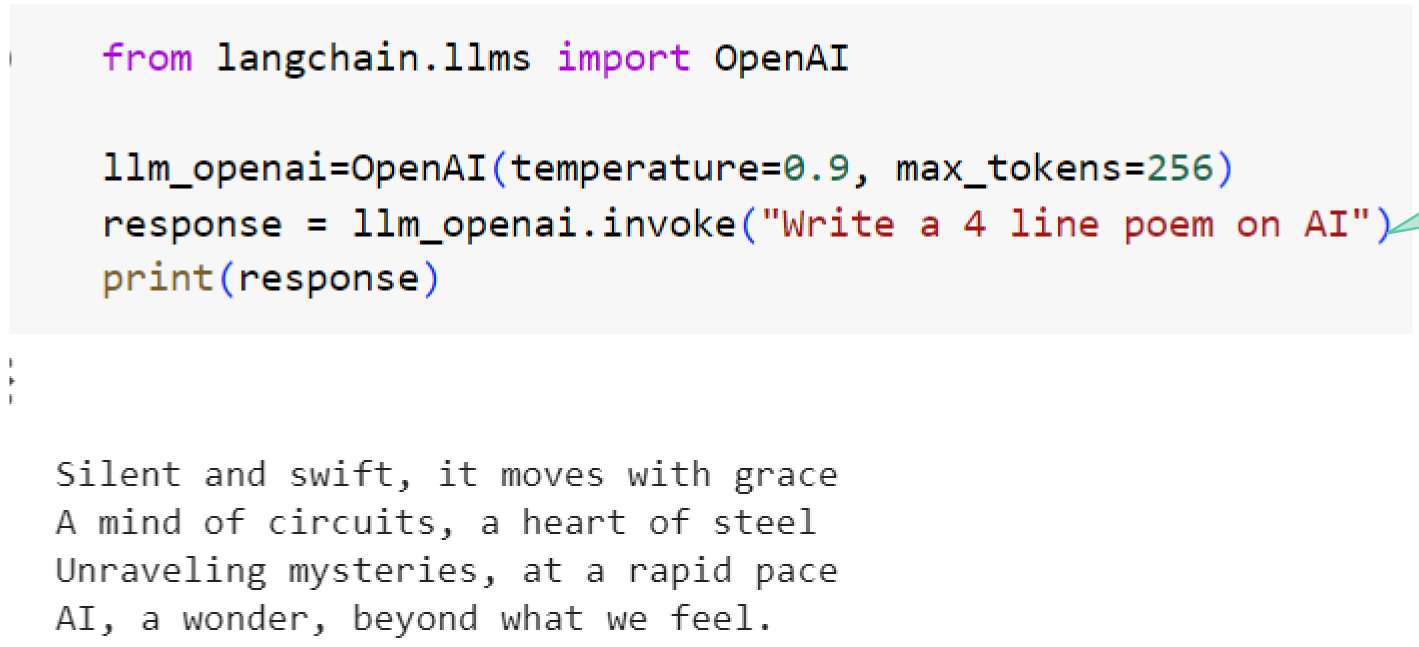
LLM Model



```python
from langchain.llms import OpenAI

llm_openai=OpenAI(temperature=0.9, max_tokens=256)
response = llm_openai.invoke("Write a 4 line poem on AI")
print(response)
```

```
Silent and swift, it moves with grace
A mind of circuits, a heart of steel
Unraveling mysteries, at a rapid pace
AI, a wonder, beyond what we feel.
```

Open Ended prompt

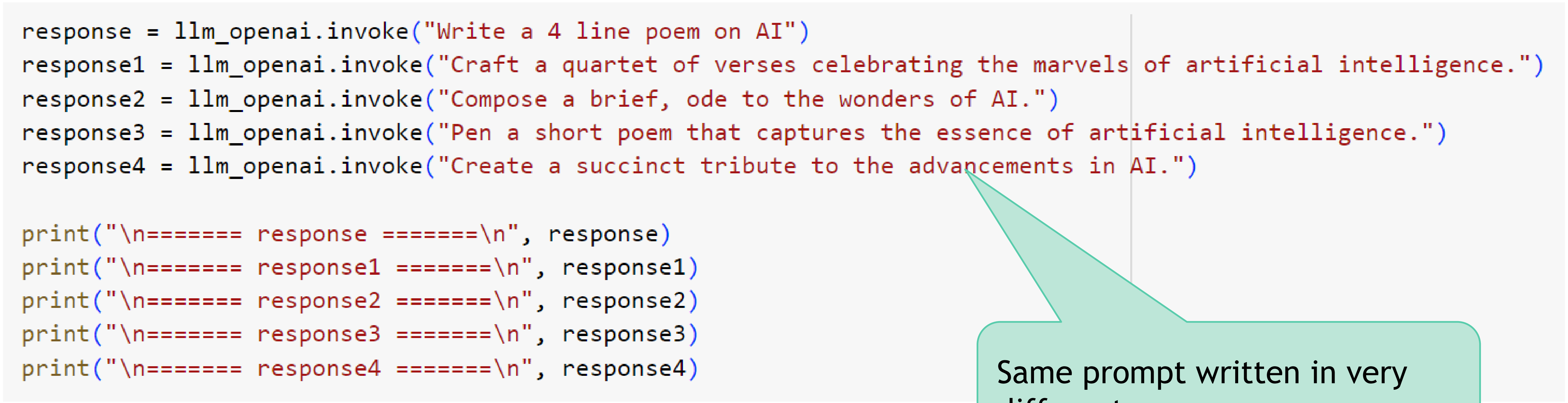


```
from langchain.llms import OpenAI

llm_openai=OpenAI(temperature=0.9, max_tokens=256)
response = llm_openai.invoke("Write a 4 line poem on AI")
print(response)
```

```
Silent and swift, it moves with grace
A mind of circuits, a heart of steel
Unraveling mysteries, at a rapid pace
AI, a wonder, beyond what we feel.
```

- This is an open ended prompt.
- User can write anything here

Open Ended prompts



```python
response = llm_openai.invoke("Write a 4 line poem on AI")
response1 = llm_openai.invoke("Craft a quartet of verses celebrating the marvels of artificial intelligence.")
response2 = llm_openai.invoke("Compose a brief, ode to the wonders of AI.")
response3 = llm_openai.invoke("Pen a short poem that captures the essence of artificial intelligence.")
response4 = llm_openai.invoke("Create a succinct tribute to the advancements in AI.")

print("\n======= response =======\n", response)
print("\n======= response1 =======\n", response1)
print("\n======= response2 =======\n", response2)
print("\n======= response3 =======\n", response3)
print("\n======= response4 =======\n", response4)
```

Same prompt written in very different ways

Open Ended prompts

```
======= response =======


Artificial intelligence, so advanced and new
A world of possibilities, it brings into view
From self-driving cars to humanoid machines
The future is here, with AI's limitless means.
```

```
======= response1 =======


Verse 1:
Oh, the wonders of AI, how it never ceases to amaze
With its algorithms and coding, it can do so many ways
From self-driving cars to virtual assistants
It's making life easier, without any resistance

Verse 2:
The speed and accuracy, it's truly a sight
As it analyzes massive data with such might
With machine learning, it continues to evolve
Solving complex problems, it's a mystery to solve
```

```
======= response2 =======


Oh wondrous AI, source of endless innovation
With each passing day, you defy imagination
Through algorithms and codes, you make our lives sublime
Your intelligence and efficiency, truly beyond time

From self-driving cars to virtual assistants
You have revolutionized our daily existence
Your ability to learn and adapt knows no bounds
Making complex tasks seem effortless, you astound
```

```
======= response4 =======


rom automating mundane tasks to revolutionizing industries,
```

If input prompts are different then the output from LLM is also very different.

Giving a structure to the prompt

- Can we restrict all the user to give certain of prompt only?
- Can we give a structure to the prompt?
- Yes- we can by using a prompt template.

Prompt Templates

- Prompt template is almost like a high level scripting language that we crate.
- There are two main reasons for using prompt templates
- **Enhanced Precision and Relevance:**
  - The use of prompt templates and limited user input phrases ensures that the LLM's responses are precisely tailored to the desired context and information needs.
- **Control Over Output Quality:**
  - By restricting user input to predefined structure, the system can maintain a high level of output quality, reducing the risk of irrelevant or off-topic responses.

Prompt Template

```
from langchain import PromptTemplate

template = "Write a 4 line poem on the subject {subject_name}"

prompt = PromptTemplate(
        input_variables=["subject_name"],
        template=template,
)

print(prompt.format(subject_name="Data Science"))
print(prompt.format(subject_name="Fathers Day"))
print(prompt.format(subject_name="Solar System"))
```

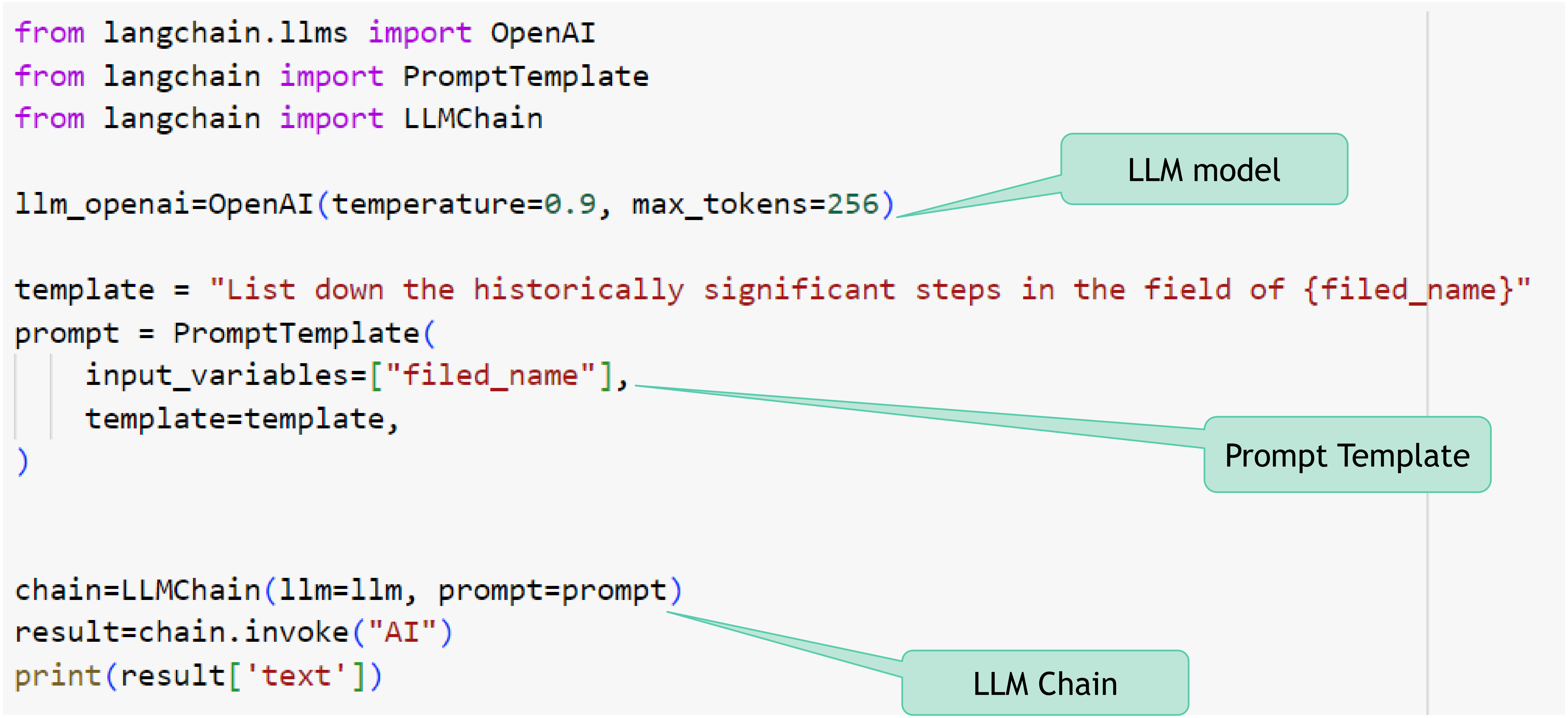
```
Write a 4 line poem on the subject Data Science
Write a 4 line poem on the subject Fathers Day
Write a 4 line poem on the subject Solar System
```

User can only change this part

LLM + Prompt Template

- We have an LLM model
- We have a prompt template
- Its time to chain together the LLM model and prompt
- We use an LLMChain() function for this operation

Finally, an LLM Chain



```python
from langchain.llms import OpenAI
from langchain import PromptTemplate
from langchain import LLMChain

llm_openai=OpenAI(temperature=0.9, max_tokens=256)

template = "List down the historically significant steps in the field of {filed_name}"
prompt = PromptTemplate(
    input_variables=["filed_name"],
    template=template,
)


chain=LLMChain(llm=llm, prompt=prompt)
result=chain.invoke("AI")
print(result['text'])
```

LAB: Example of an LLM Chain

- Create a chain for the below context.
  - Explain any topic to a layman
  - For example the topic name is *Machine Learning*. Explain this topic to a layman

LAB: Example of an LLM Chain

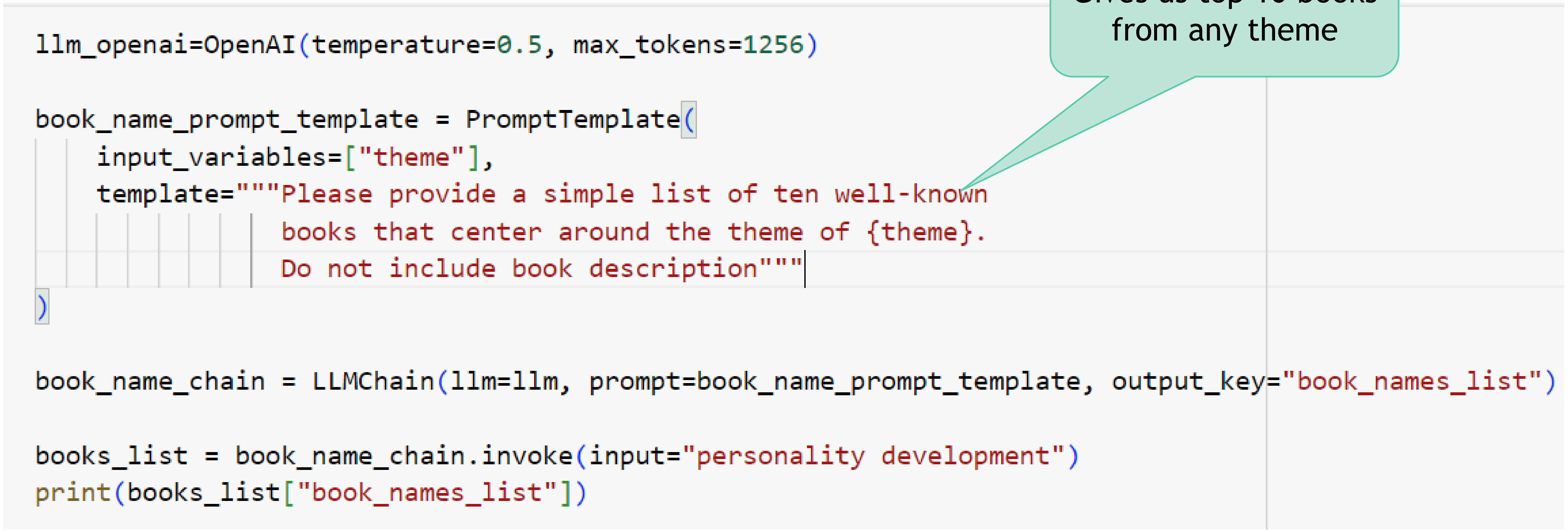
```
llm_openai=OpenAI(temperature=0.9, max_tokens=1256)

template = "The topic name is {topic}. Explain this topic to a 10 years old kid"
prompt = PromptTemplate(
    input_variables=["topic"],
    template=template,
)

chain=LLMChain(llm=llm, prompt=prompt)
result=chain.invoke("Logistic Regression")
print(result['text'])
```

Sequential Chain

Sequential Chain

- Helps us in connecting two chains
- Input of first chain will be output of second chain

Chain-1 : Gets top 10 books

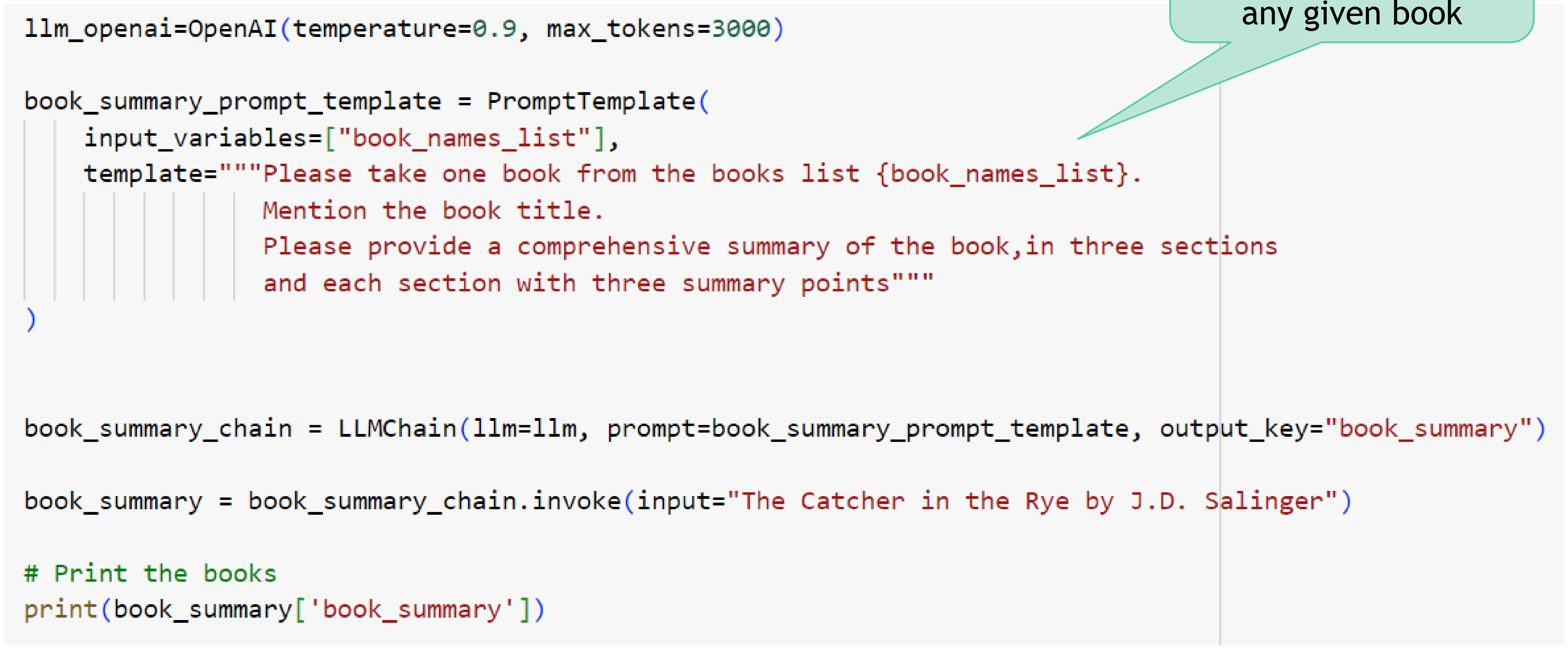


```
llm_openai=OpenAI(temperature=0.5, max_tokens=1256)

book_name_prompt_template = PromptTemplate(
    input_variables=["theme"],
    template="""Please provide a simple list of ten well-known
            books that center around the theme of {theme}.
            Do not include book description"""
)

book_name_chain = LLMChain(llm=llm, prompt=book_name_prompt_template, output_key="book_names_list")

books_list = book_name_chain.invoke(input="personality development")
print(books_list["book_names_list"])
```

Gives us top 10 books from any theme

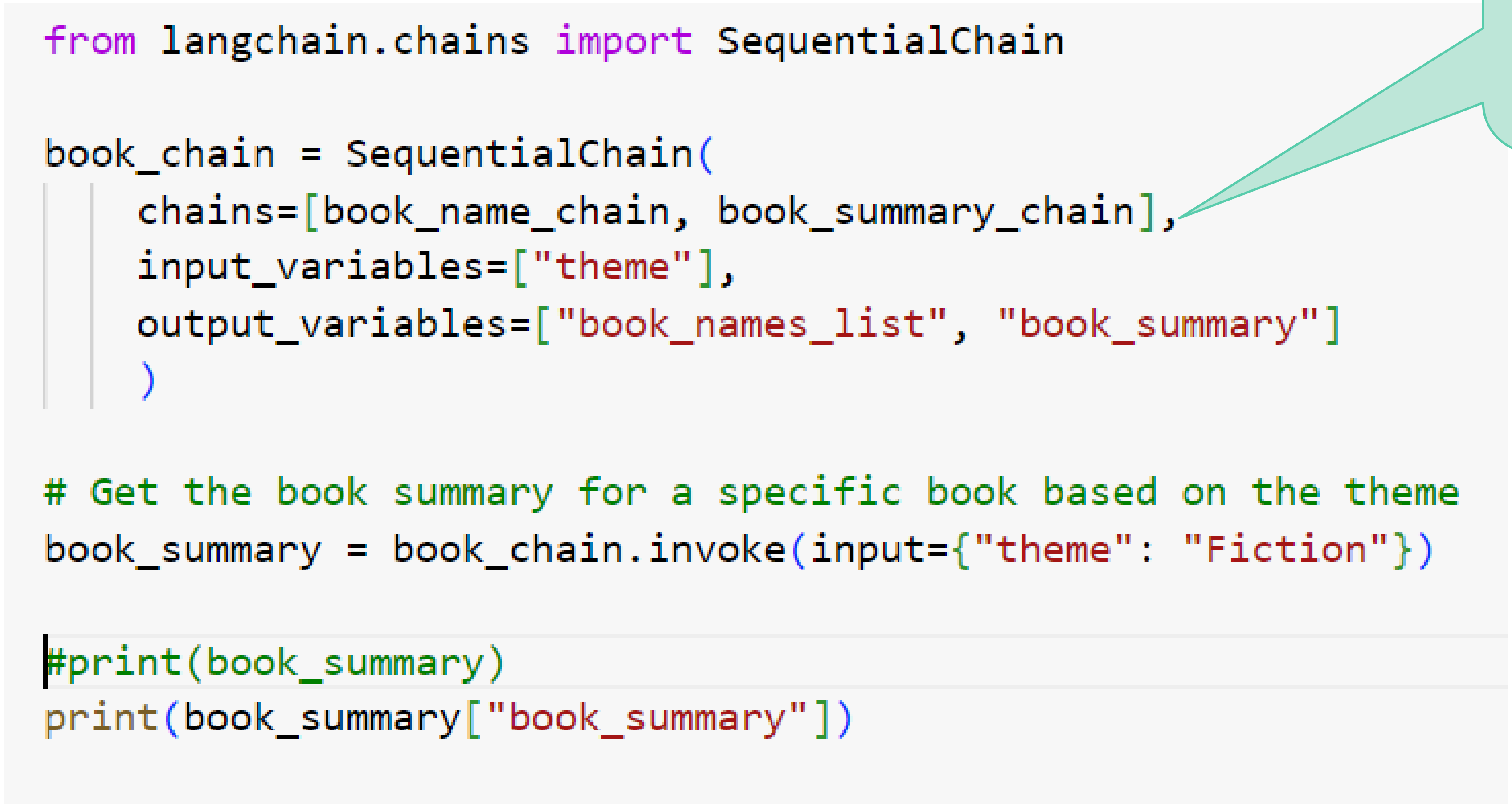
Chain-2: Gives the summary of a book

This chain provides us the summary of any given book



```python
llm_openai=OpenAI(temperature=0.9, max_tokens=3000)

book_summary_prompt_template = PromptTemplate(
    input_variables=["book_names_list"],
    template="""Please take one book from the books list {book_names_list}.
            Mention the book title.
            Please provide a comprehensive summary of the book,in three sections
            and each section with three summary points"""
)


book_summary_chain = LLMChain(llm=llm, prompt=book_summary_prompt_template, output_key="book_summary")

book_summary = book_summary_chain.invoke(input="The Catcher in the Rye by J.D. Salinger")

# Print the books
print(book_summary['book_summary'])
```

Sequential Chain



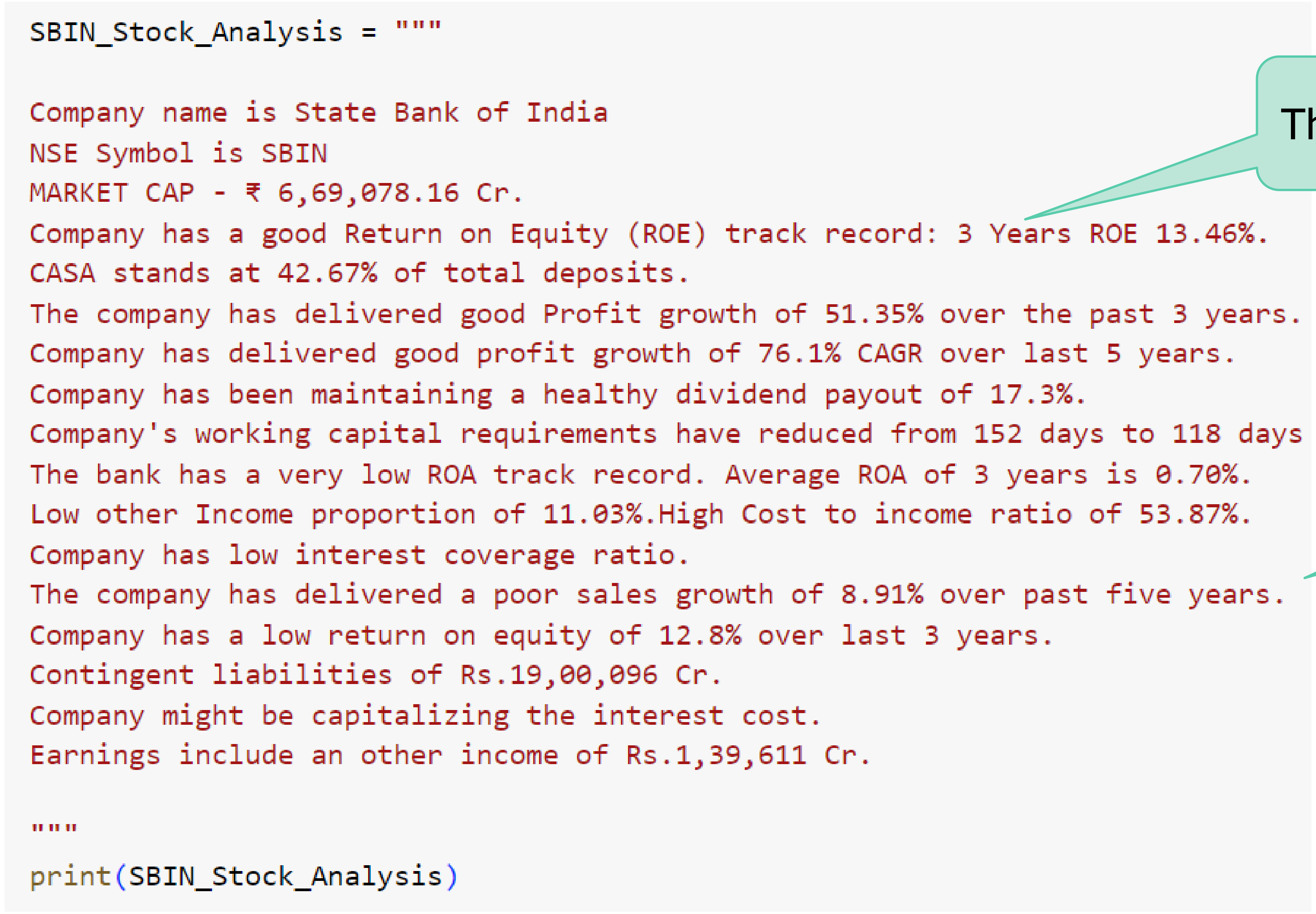
```python
from langchain.chains import SequentialChain

book_chain = SequentialChain(
        chains=[book_name_chain, book_summary_chain],
        input_variables=["theme"],
        output_variables=["book_names_list", "book_summary"]
        )

# Get the book summary for a specific book based on the theme
book_summary = book_chain.invoke(input={"theme": "Fiction"})

#print(book_summary)
print(book_summary["book_summary"])
```

a sequential chain that first gets the book names based on the theme and then gets the summary of a specific book

LAB: Sequential Chain

- Take the data from a stock analysis report.
- Create a chain to decide whether to “BUY” or “HOLD” or “SELL” the stock
- Follow these three steps
  - Create a chain that takes the “stock analysis report” as input and gives the “positives and negatives” about the stock as output
  - Create a chain that takes the “positives and negatives” about the stock as input and prepares an “investor report” on whether buying that stock is a good option or not
  - Create a chain that takes the “investor report” as input and gives us “BUY” or “HOLD” or “SELL” signal

LAB: Sequential Chain

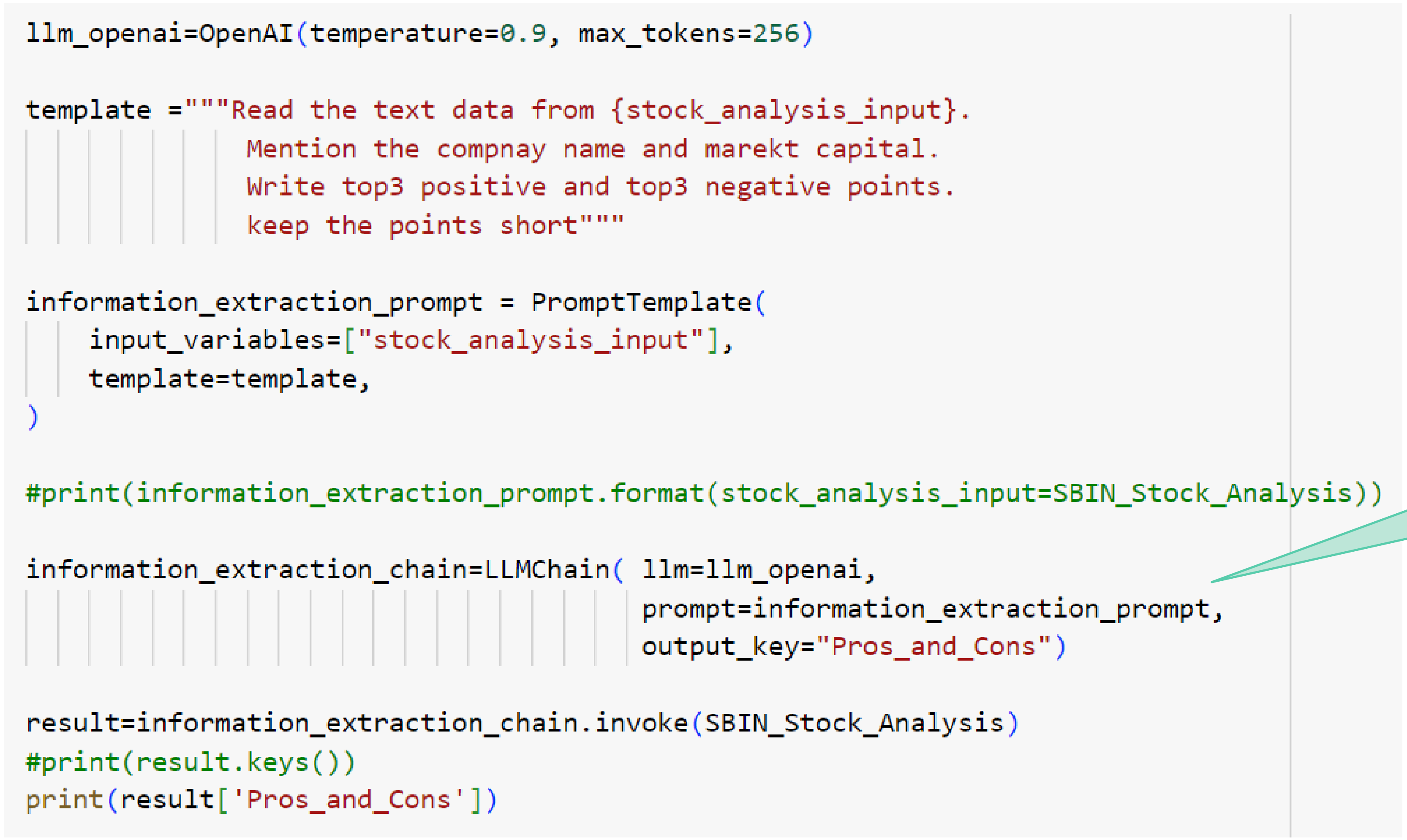


```
SBIN_Stock_Analysis = """

Company name is State Bank of India
NSE Symbol is SBIN
MARKET CAP - ₹ 6,69,078.16 Cr.
Company has a good Return on Equity (ROE) track record: 3 Years ROE 13.46%.
CASA stands at 42.67% of total deposits.
The company has delivered good Profit growth of 51.35% over the past 3 years.
Company has delivered good profit growth of 76.1% CAGR over last 5 years.
Company has been maintaining a healthy dividend payout of 17.3%.
Company's working capital requirements have reduced from 152 days to 118 days
The bank has a very low ROA track record. Average ROA of 3 years is 0.70%.
Low other Income proportion of 11.03%.High Cost to income ratio of 53.87%.
Company has low interest coverage ratio.
The company has delivered a poor sales growth of 8.91% over past five years.
Company has a low return on equity of 12.8% over last 3 years.
Contingent liabilities of Rs.19,00,096 Cr.
Company might be capitalizing the interest cost.
Earnings include an other income of Rs.1,39,611 Cr.

"""
print(SBIN_Stock_Analysis)
```

The input data

We will use document loaders in the later sessions

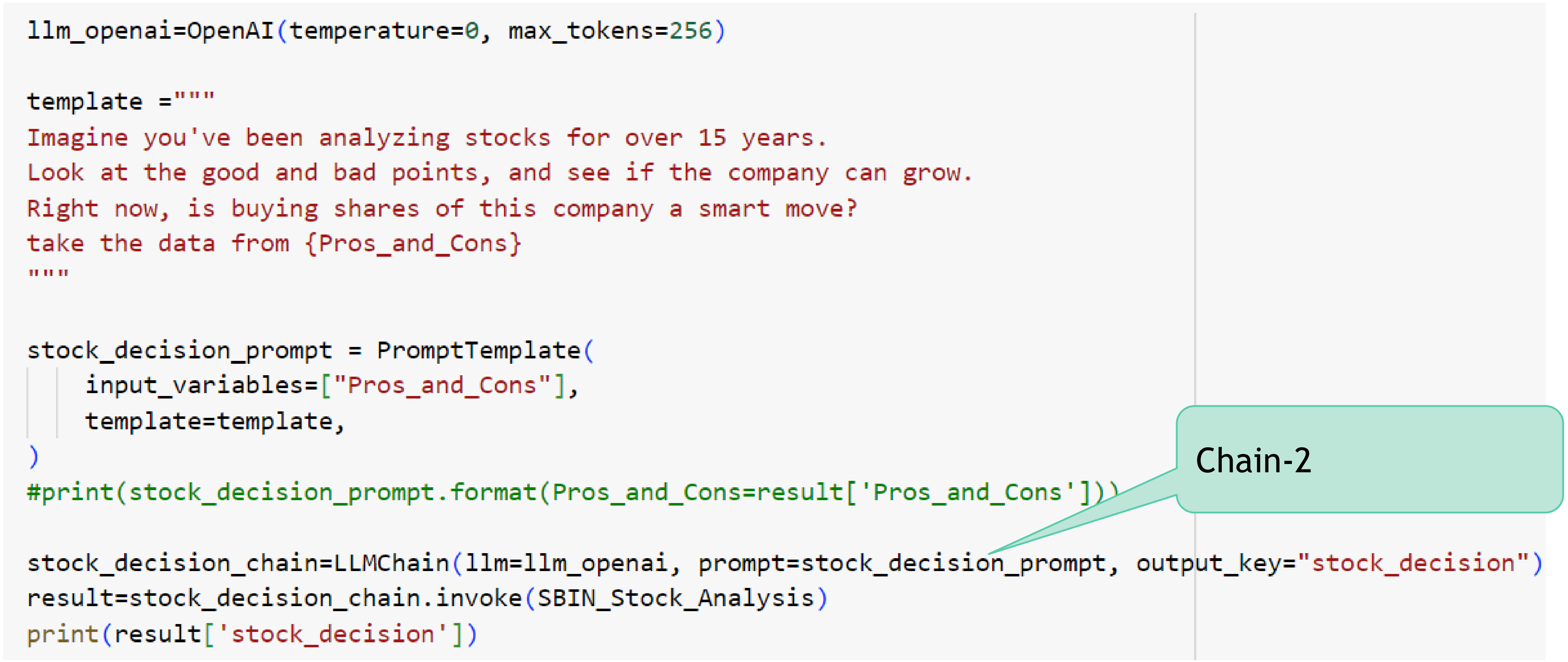
LAB: Sequential Chain



```
llm_openai=OpenAI(temperature=0.9, max_tokens=256)

template ="""Read the text data from {stock_analysis_input}.
              Mention the compnay name and marekt capital.
              Write top3 positive and top3 negative points.
              keep the points short"""

information_extraction_prompt = PromptTemplate(
    input_variables=["stock_analysis_input"],
    template=template,
)

#print(information_extraction_prompt.format(stock_analysis_input=SBIN_Stock_Analysis))

information_extraction_chain=LLMChain( llm=llm_openai,
                                       prompt=information_extraction_prompt,
                                       output_key="Pros_and_Cons")

result=information_extraction_chain.invoke(SBIN_Stock_Analysis)
#print(result.keys())
print(result['Pros_and_Cons'])
```

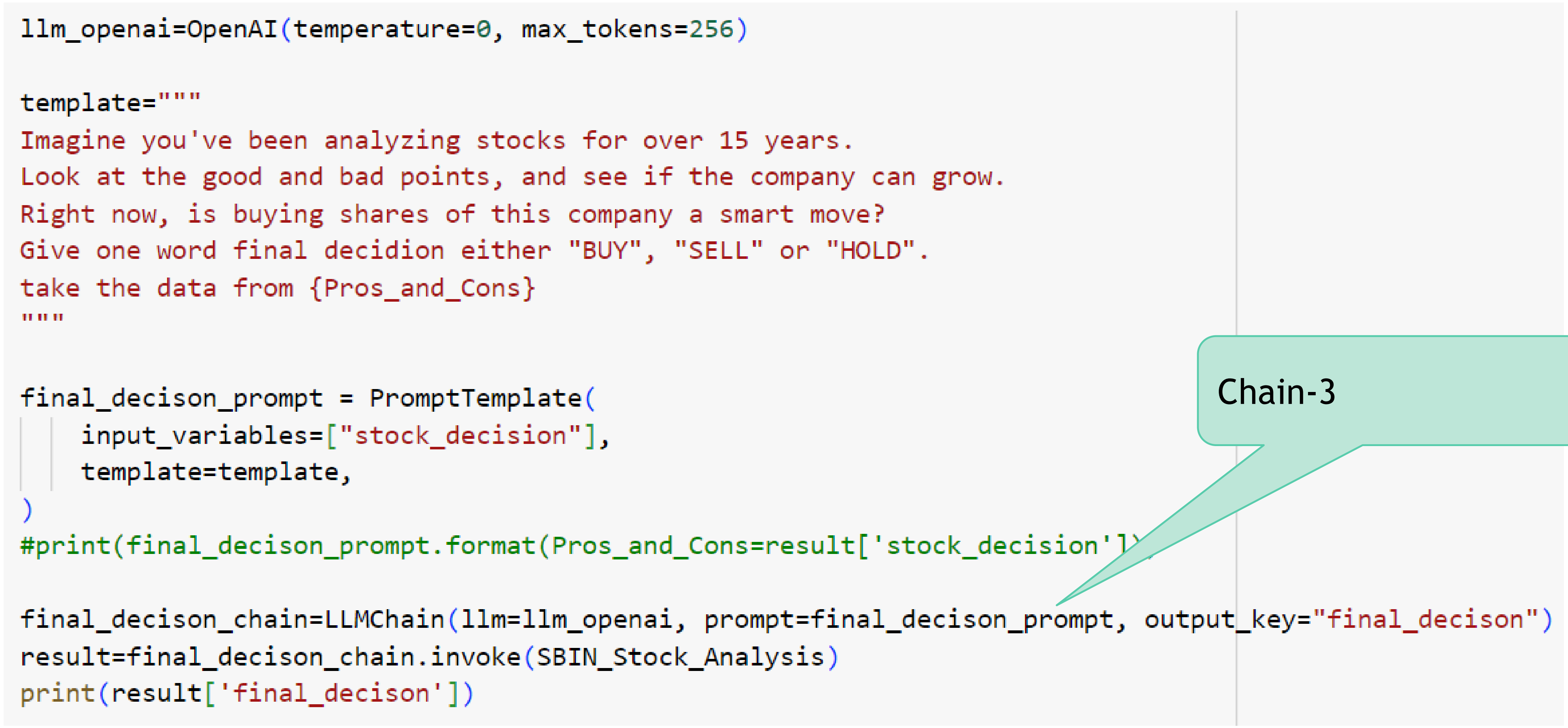
Chain-1

LAB: Sequential Chain



```
llm_openai=OpenAI(temperature=0, max_tokens=256)

template ="""
Imagine you've been analyzing stocks for over 15 years.
Look at the good and bad points, and see if the company can grow.
Right now, is buying shares of this company a smart move?
take the data from {Pros_and_Cons}
"""

stock_decision_prompt = PromptTemplate(
    input_variables=["Pros_and_Cons"],
    template=template,
)
#print(stock_decision_prompt.format(Pros_and_Cons=result['Pros_and_Cons']))

stock_decision_chain=LLMChain(llm=llm_openai, prompt=stock_decision_prompt, output_key="stock_decision")
result=stock_decision_chain.invoke(SBIN_Stock_Analysis)
print(result['stock_decision'])
```

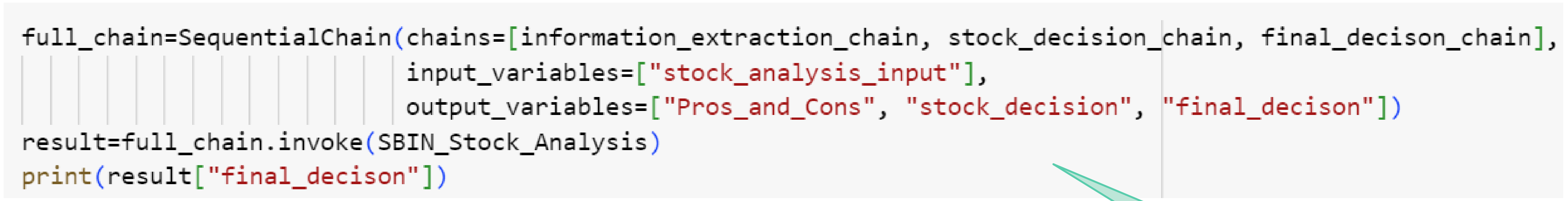
LAB: Sequential Chain



```
llm_openai=OpenAI(temperature=0, max_tokens=256)

template="""
Imagine you've been analyzing stocks for over 15 years.
Look at the good and bad points, and see if the company can grow.
Right now, is buying shares of this company a smart move?
Give one word final decidion either "BUY", "SELL" or "HOLD".
take the data from {Pros_and_Cons}
"""

final_decison_prompt = PromptTemplate(
    input_variables=["stock_decision"],
    template=template,
)
#print(final_decison_prompt.format(Pros_and_Cons=result['stock_decision']))

final_decison_chain=LLMChain(llm=llm_openai, prompt=final_decison_prompt, output_key="final_decison")
result=final_decison_chain.invoke(SBIN_Stock_Analysis)
print(result['final_decison'])
```

Chain-3

LAB: Sequential Chain



```
full_chain=SequentialChain(chains=[information_extraction_chain, stock_decision_chain, final_decison_chain],
                           input_variables=["stock_analysis_input"],
                           output_variables=["Pros_and_Cons", "stock_decision", "final_decison"])
result=full_chain.invoke(SBIN_Stock_Analysis)
print(result["final_decison"])
```

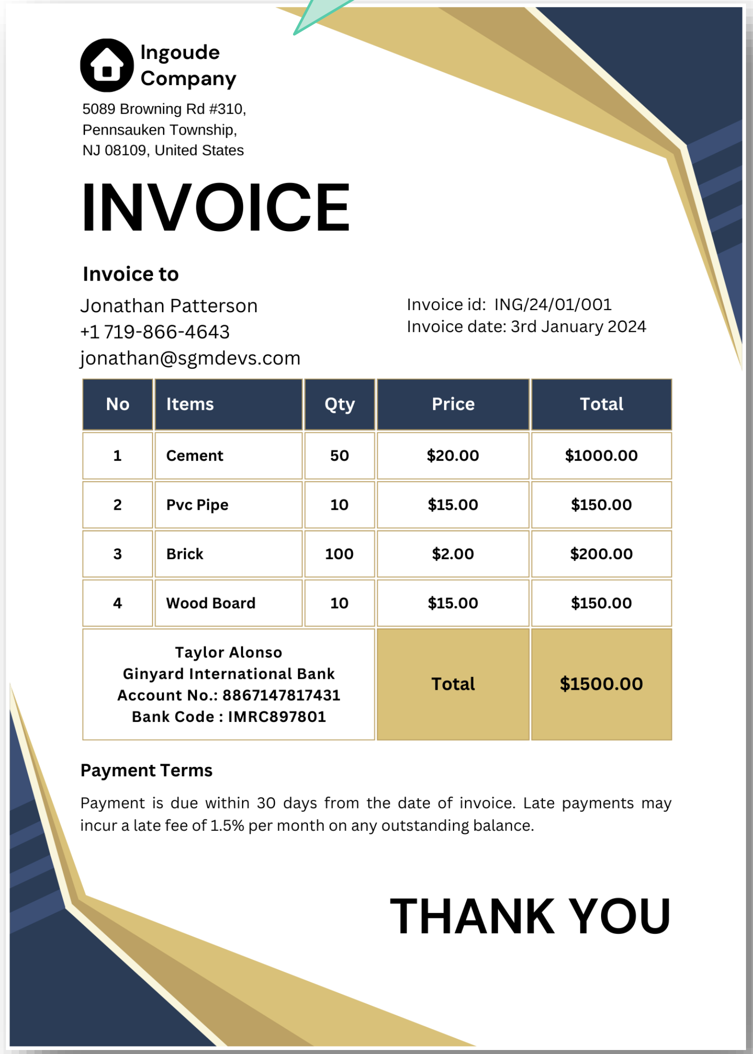
Final Sequential Chain

IDP (Intelligent Document Processing)

- Intelligent Document Processing (IDP) leverages advanced technologies to automatically process, analyze, and manage documents.
- IDP systems can handle various formats and types of unstructured or semi-structured data, transforming them into structured, actionable information.
- These technologies aim to improve efficiency, accuracy, and the speed of data processing tasks.

DV Analytics
Transforming You

Invoice document in an Image



Ingoude Company
5089 Browning Rd #310,
Pennsauken Township,
NJ 08109, United States

INVOICE

Invoice to
Jonathan Patterson
+1 719-866-4643
jonathan@sgmdevs.com

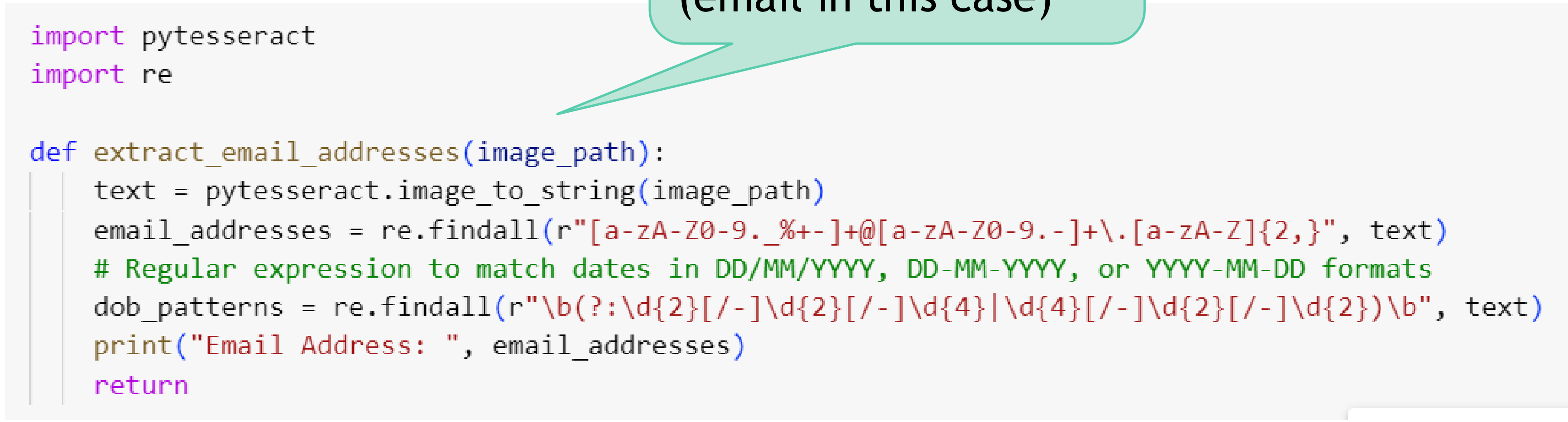
Invoice id: ING/24/01/001
Invoice date: 3rd January 2024

No	Items	Qty	Price	Total
1	Cement	50	$20.00	$1000.00
2	Pvc Pipe	10	$15.00	$150.00
3	Brick	100	$2.00	$200.00
4	Wood Board	10	$15.00	$150.00
Taylor Alonso Ginyard International Bank Account No.: 8867147817431 Bank Code : IMRC897801			Total	$1500.00

Payment Terms
Payment is due within 30 days from the date of invoice. Late payments may incur a late fee of 1.5% per month on any outstanding balance.

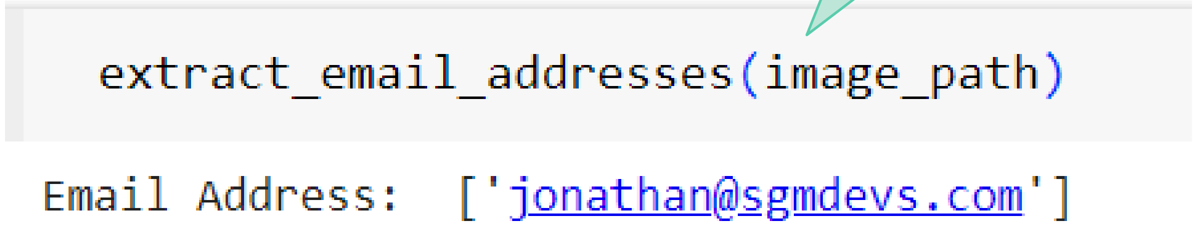
THANK YOU

Function to extract a specific information (email in this case)



```python
import pytesseract
import re

def extract_email_addresses(image_path):
    text = pytesseract.image_to_string(image_path)
    email_addresses = re.findall(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", text)
    # Regular expression to match dates in DD/MM/YYYY, DD-MM-YYYY, or YYYY-MM-DD formats
    dob_patterns = re.findall(r"\b(?:\d{2}[/-]\d{2}[/-]\d{4}|\d{4}[/-]\d{2}[/-]\d{2})\b", text)
    print("Email Address: ", email_addresses)
    return
```

Result



```
extract_email_addresses(image_path)
```

```
Email Address:  ['jonathan@sgmdevs.com']
```

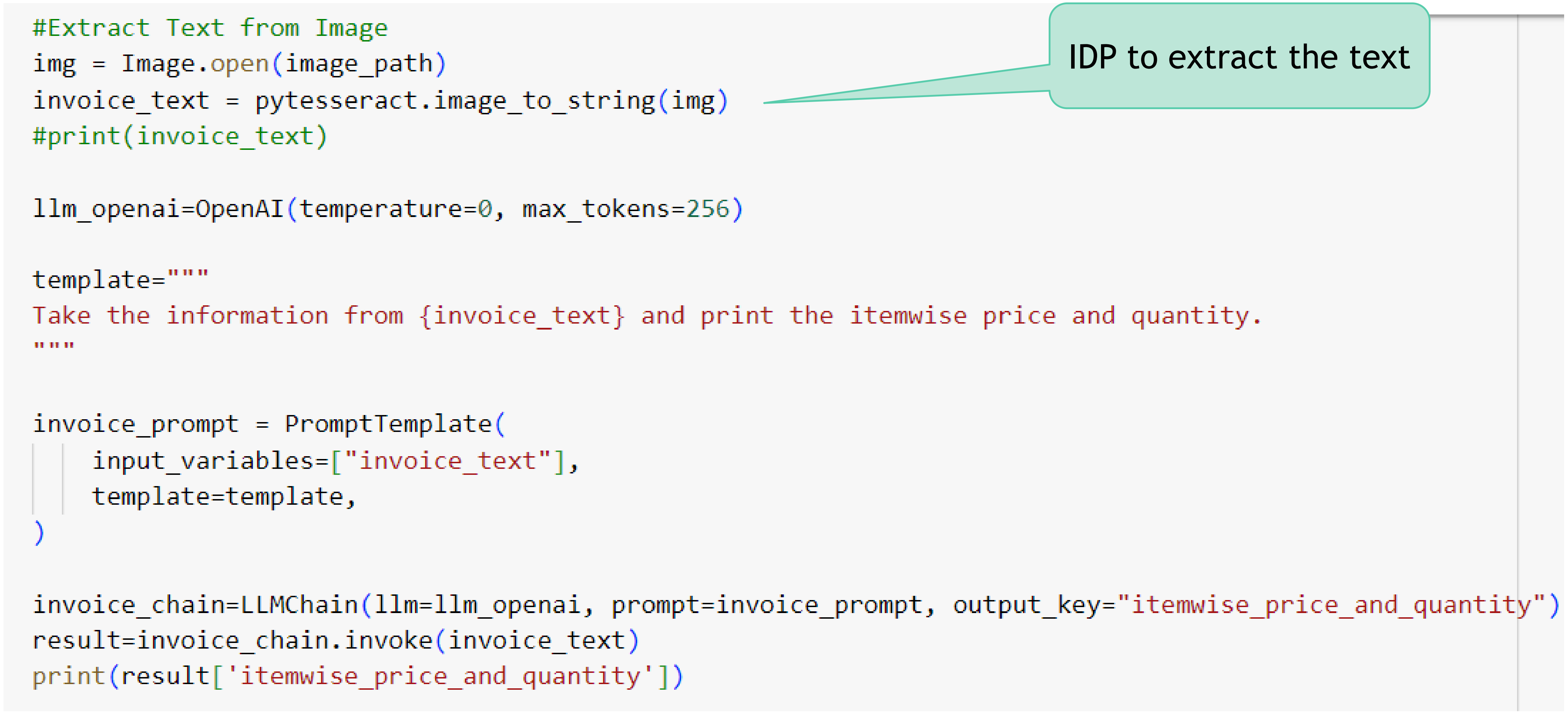
Examples of IDP Applications

- Invoice Processing
  - Automatically extracting data from invoices such as vendor details, amounts, and dates, then inputting this data into financial systems.
- Customer Onboarding
  - Streamlining the process of gathering and verifying documents for new customer accounts in banking or telecom.
- Claims Processing
  - In the insurance sector, IDP can automate the extraction and analysis of claim documents to facilitate faster claim resolution.
- Email and Attachment Processing
  - Automatically extracting and processing relevant information from emails and their attachments for further action or analysis.
- Medical Records Management
  - Extracting patient information from unstructured medical records to support diagnoses, treatment plans, and billing.

LLM + IDP

- If we only use IDP, we need to write many codes about regular expressions to get the information we want.
- If the documents don't match well, it's hard to get the information.
- By using LLMs with IDP, we can get more information with less code.
- We can combine the strengths of LLMs and IDP.

Extract the table

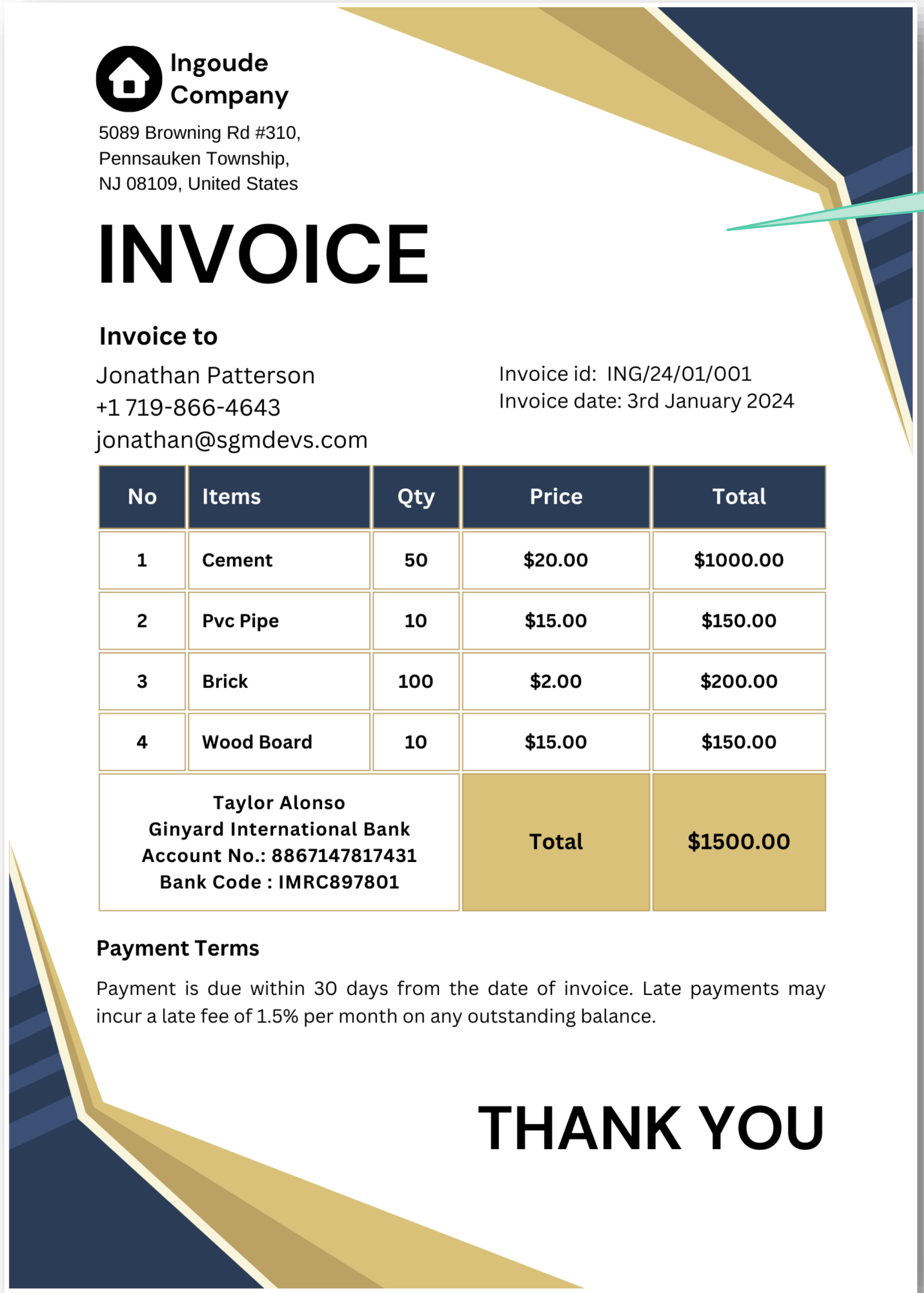


```
#Extract Text from Image
img = Image.open(image_path)
invoice_text = pytesseract.image_to_string(img)
#print(invoice_text)

llm_openai=OpenAI(temperature=0, max_tokens=256)

template="""
Take the information from {invoice_text} and print the itemwise price and quantity.
"""


invoice_prompt = PromptTemplate(
    input_variables=["invoice_text"],
    template=template,
)

invoice_chain=LLMChain(llm=llm_openai, prompt=invoice_prompt, output_key="itemwise_price_and_quantity")
result=invoice_chain.invoke(invoice_text)
print(result['itemwise_price_and_quantity'])
```

IDP to extract the text

DV Analytics
Transforming You

Input



Ingoude Company

5089 Browning Rd #310,
Pennsauken Township,
NJ 08109, United States

INVOICE

Invoice to

Jonathan Patterson
+1 719-866-4643
jonathan@sgmdevs.com

Invoice id: ING/24/01/001
Invoice date: 3rd January 2024

No	Items	Qty	Price	Total
1	Cement	50	$20.00	$1000.00
2	Pvc Pipe	10	$15.00	$150.00
3	Brick	100	$2.00	$200.00
4	Wood Board	10	$15.00	$150.00
Taylor Alonso Ginyard International Bank Account No.: 8867147817431 Bank Code : IMRC897801			Total	$1500.00

Payment Terms

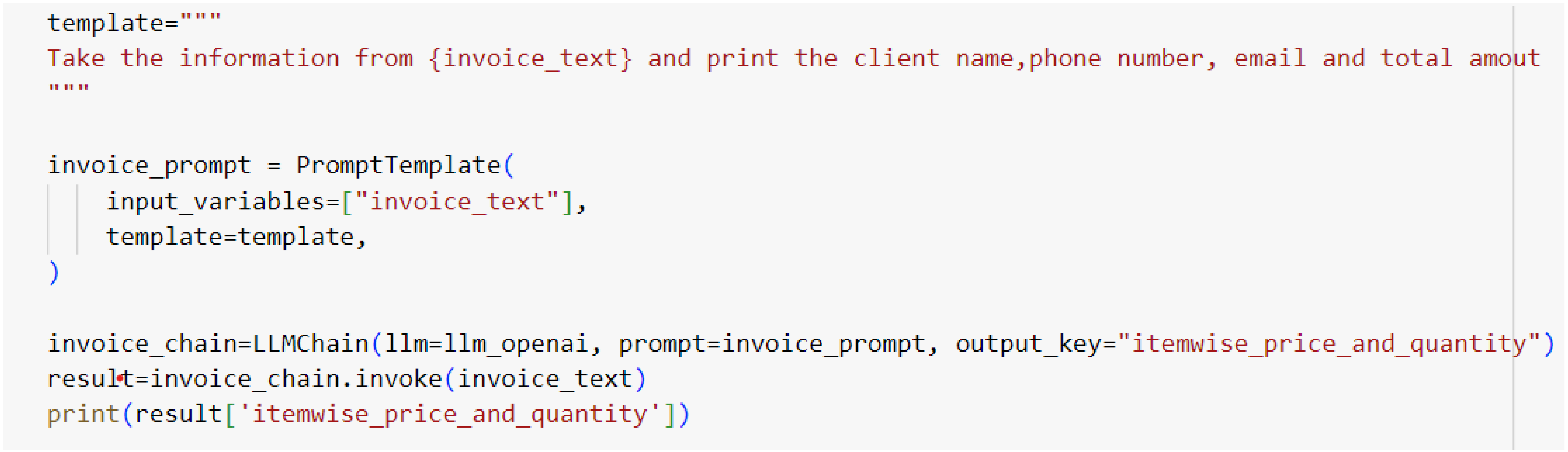
Payment is due within 30 days from the date of invoice. Late payments may incur a late fee of 1.5% per month on any outstanding balance.

THANK YOU

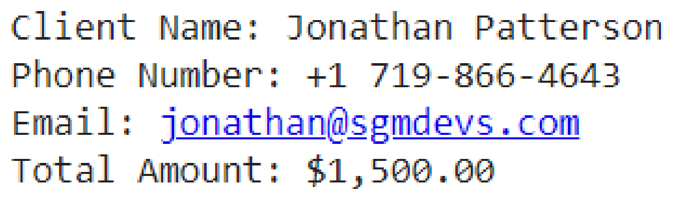
```
Item | Quantity | Price | Total
--- | --- | --- | ---
Cement | 1000 | $20.00 | $20,000.00
Pvc Pipe | 10 | $15.00 | $150.00
Brick | 100 | $2.00 | $200.00
Wood Board | 10 | $15.00 | $150.00
```

Output

Extract Client name and Specific Details



```
template="""
Take the information from {invoice_text} and print the client name,phone number, email and total amout
"""


invoice_prompt = PromptTemplate(
    input_variables=["invoice_text"],
    template=template,
)

invoice_chain=LLMChain(llm=llm_openai, prompt=invoice_prompt, output_key="itemwise_price_and_quantity")
result=invoice_chain.invoke(invoice_text)
print(result['itemwise_price_and_quantity'])
```



```
Client Name: Jonathan Patterson
Phone Number: +1 719-866-4643
Email: jonathan@sgmdevs.com
Total Amount: $1,500.00
```

Better results with very less lines of code

Assignment :
Book Summary App

Assignment : Book Summary App

- Develop an application using LangChain's sequential chain feature.
- The application should be structured as follows:
  - The first chain in the sequence will accept a theme as input and identify the top 10 books related to that theme.
  - The second chain will then generate a summary for each of these books.

Book Summary App

Book Summary App – Code Files

- Colab File
  - https://colab.research.google.com/drive/1IomOBIfvpONcgVroiAQW6xKzZCZNoPKm?pli=1&usp=drive_fs
- Hugging Face
  - https://huggingface.co/spaces/venkatareddykonasani/Book_Summary

DV Analytics
Transforming You

Thank you