In [1]:

```

```

In [ ]:

```python
#Tools_Agents_project_1
```

In [345…

```python
#!pip install --upgrade langchain langchain-core langchain-community langchain-groq groq
```

In [ ]:

```python
# !pip install --upgrade langchain
```

In [ ]:

```python
#!pip install --upgrade langchain-core
```

In [ ]:

```python
#!pip install --upgrade langchain-community
```

In [ ]:

```python
#!pip install --upgrade langchain-groq groq
```

In [347…

```python
# load environment variables from the ".env" file.

import os
from dotenv import load_dotenv
load_dotenv()
```

Out[347…

```
True
```

In [349…

```python
#  import WikipediaQueryRun

from langchain_community.tools import WikipediaQueryRun
```

In [351…

```python
#  import WikipediaAPIWrapper

from langchain_community.utilities import WikipediaAPIWrapper
```

In [353…

```python
# Initialize a WikipediaAPIWrapper with top_k_results, and doc_content_chars_max.

wikipedia_api_wrapper=WikipediaAPIWrapper(top_k_results=3,doc_content_chars_max=500, lang = "en")

# use wikipedia_api_wrapper into WikipediaQueryRun
wikiquery=WikipediaQueryRun(api_wrapper=wikipedia_api_wrapper)
```

In [355…

```python
# import "OPENAI_API_KEY"

os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")
```

In [357…

```python
from langchain_openai import OpenAIEmbeddings

from langchain_community.vectorstores import FAISS
```

In [359…

```python
from langchain_community.document_loaders import WebBaseLoader
webbase_loader=WebBaseLoader("https://python.langchain.com/docs/introduction/")

doc_web=webbase_loader.load()
```

In [361…

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter

doc_rec=RecursiveCharacterTextSplitter(chunk_size=500,chunk_overlap=60).split_documents(doc_web)

vector_store=FAISS.from_documents(doc_rec, OpenAIEmbeddings())
retriever_data=vector_store.as_retriever()
```

In [365…

```python
from langchain.tools.retriever import create_retriever_tool
```

In [367…

```python
# Generate retriever
retriv = vector_store.as_retriever()

# Define a function with user input

def retriv_fun(user_input):
    return retriv.invoke(user_input)
```

In [373…

```python
retriever_tool_fun = Tool.from_function(
    func=retriv_fun,
    name="retriever_work_test",
    description="retriever performance test."
)
```

In [375…

```python
tools=[wikiquery,retriever_tool_fun]
```

In [377…

```python
tools
```

Out[377…

```
[WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper(wiki_client=<module 'wikipedia' from 'C:\\Users\\gobin\\anaconda3\\Lib\\site-packages\\wikipedia\\__init__.py'>, top_k_results=3, lang='en', load_all_available_meta=False, doc_content_chars_max=500)),
 Tool(name='retriever_work_test', description='retriever performance test.', func=<function retriv_fun at 0x000001A4902160C0>)]
```

In [379…

```python
# import "GROQ API key"

os.environ["GROQ_API_KEY"] = os.getenv("GROQ_API_KEY")
```

In [381…

```python
# load ChatGroq
from langchain_groq import ChatGroq
```

In [383…

```python
# Initialize the ChatGroq,  ChatOpenAI ()

#llm = ChatGroq(model="llama3-8b-8192")

from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4-turbo")
```

In [385…

```python
# query Template with LangChain hub


# load LangChain Hub
from langchain import hub

query=hub.pull("hwchase17/openai-functions-agent")
```

In [387…

```python
# Make an agent
from langchain.agents import create_openai_tools_agent

# Give theree parameters to make agent.

agent_3_pars=create_openai_tools_agent(llm, tools,query)
agent_3_pars
```

Out[387…

```
RunnableAssign(mapper={
  agent_scratchpad: RunnableLambda(lambda x: format_to_openai_tool_messages(x['intermediate_steps']))
})
| ChatPromptTemplate(input_variables=['agent_scratchpad', 'input'], optional_variables=['chat_history'], input_types={'chat_history': list[typing.Annotated[typing.Union[typing.Annotated[langchain_core.messages.ai.AIMessage, Tag(tag='ai')], typing.Annotated[langchain_core.messages.human.HumanMessage, Tag(tag='human')], typing.Annotated[langchain_core.messages.chat.ChatMessage, Tag(tag='chat')], typing.Annotated[langchain_core.messages.system.SystemMessage, Tag(tag='system')], typing.Annotated[langchain_core.messages.function.FunctionMessage, Tag(tag='function')], typing.Annotated[langchain_core.messages.tool.ToolMessage, Tag(tag='tool')], typing.Annotated[langchain_core.messages.ai.AIMessageChunk, Tag(tag='AIMessageChunk')], typing.Annotated[langchain_core.messages.human.HumanMessageChunk, Tag(tag='HumanMessageChunk')], typing.Annotated[langchain_core.messages.chat.ChatMessageChunk, Tag(tag='ChatMessageChunk')], typing.Annotated[langchain_core.messages.system.SystemMessageChunk, Tag(tag='SystemMessageChunk')], typing.Annotated[langchain_core.messages.function.FunctionMessageChunk, Tag(tag='FunctionMessageChunk')], typing.Annotated[langchain_core.messages.tool.ToolMessageChunk, Tag(tag='ToolMessageChunk')]], FieldInfo(annotation=NoneType, required=True, discriminator=Discriminator(discriminator=<function _get_type at 0x000001A4719A2C00>, custom_error_type=None, custom_error_message=None, custom_error_context=None))]], 'agent_scratchpad': list[typing.Annotated[typing.Union[typing.Annotated[langchain_core.messages.ai.AIMessage, Tag(tag='ai')], typing.Annotated[langchain_core.messages.human.HumanMessage, Tag(tag='human')], typing.Annotated[langchain_core.messages.chat.ChatMessage, Tag(tag='chat')], typing.Annotated[langchain_core.messages.system.SystemMessage, Tag(tag='system')], typing.Annotated[langchain_core.messages.function.FunctionMessage, Tag(tag='function')], typing.Annotated[langchain_core.messages.tool.ToolMessage, Tag(tag='tool')], typing.Annotated[langchain_core.messages.ai.AIMessageChunk, Tag(tag='AIMessageChunk')], typing.Annotated[langchain_core.messages.human.HumanMessageChunk, Tag(tag='HumanMessageChunk')], typing.Annotated[langchain_core.messages.chat.ChatMessageChunk, Tag(tag='ChatMessageChunk')], typing.Annotated[langchain_core.messages.system.SystemMessageChunk, Tag(tag='SystemMessageChunk')], typing.Annotated[langchain_core.messages.function.FunctionMessageChunk, Tag(tag='FunctionMessageChunk')], typing.Annotated[langchain_core.messages.tool.ToolMessageChunk, Tag(tag='ToolMessageChunk')]], FieldInfo(annotation=NoneType, required=True, discriminator=Discriminator(discriminator=<function _get_type at 0x000001A4719A2C00>, custom_error_type=None, custom_error_message=None, custom_error_context=None))]]}, partial_variables={'chat_history': []}, metadata={'lc_hub_owner': 'hwchase17', 'lc_hub_repo': 'openai-functions-agent', 'lc_hub_commit_hash': 'a1655024b06afbd95d17449f21316291e0726f13dcfaf990cc0d18087ad689a5'}, messages=[SystemMessagePromptTemplate(prompt=PromptTemplate(input_variables=[], input_types={}, partial_variables={}, template='You are a helpful assistant'), additional_kwargs={}), MessagesPlaceholder(variable_name='chat_history', optional=True), HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['input'], input_types={}, partial_variables={}, template='{input}'), additional_kwargs={}), MessagesPlaceholder(variable_name='agent_scratchpad')])
| RunnableBinding(bound=ChatOpenAI(client=<openai.resources.chat.completions.completions.Completions object at 0x000001A49035F890>, async_client=<openai.resources.chat.completions.completions.AsyncCompletions object at 0x000001A49035FBC0>, root_client=<openai.OpenAI object at 0x000001A49035FFB0>, root_async_client=<openai.AsyncOpenAI object at 0x000001A49035F0E0>, model_name='gpt-4-turbo', model_kwargs={}, openai_api_key=SecretStr('**********')), kwargs={'tools': [{'type': 'function', 'function': {'name': 'wikipedia', 'description': 'A wrapper around Wikipedia. Useful for when you need to answer general questions about people, places, companies, facts, historical events, or other subjects. Input should be a search query.', 'parameters': {'properties': {'query': {'description': 'query to look up on wikipedia', 'type': 'string'}}, 'required': ['query'], 'type': 'object'}}}, {'type': 'function', 'function': {'name': 'retriever_work_test', 'description': 'retriever performance test.', 'parameters': {'properties': {'__arg1': {'title': '__arg1', 'type': 'string'}}, 'required': ['__arg1'], 'type': 'object'}}}]}, config={}, config_factories=[])
| OpenAIToolsAgentOutputParser()
```

In [389…

```python
## Agent Executer

# Load AgentExecutor
from langchain.agents import AgentExecutor

# Inatialize AgentExecutor (agent, tools, and verboe)

agent_executor_results=AgentExecutor(agent=agent_3_pars,tools= agent_tools ,verbose=True)

agent_executor_results
```

Out[389…

```
AgentExecutor(verbose=True, agent=RunnableMultiActionAgent(runnable=RunnableAssign(mapper={
  agent_scratchpad: RunnableLambda(lambda x: format_to_openai_tool_messages(x['intermediate_steps']))
})
| ChatPromptTemplate(input_variables=['agent_scratchpad', 'input'], optional_variables=['chat_history'], input_types={'chat_history': list[typing.Annotated[typing.Union[typing.Annotated[langchain_core.messages.ai.AIMessage, Tag(tag='ai')], typing.Annotated[langchain_core.messages.human.HumanMessage, Tag(tag='human')], typing.Annotated[langchain_core.messages.chat.ChatMessage, Tag(tag='chat')], typing.Annotated[langchain_core.messages.system.SystemMessage, Tag(tag='system')], typing.Annotated[langchain_core.messages.function.FunctionMessage, Tag(tag='function')], typing.Annotated[langchain_core.messages.tool.ToolMessage, Tag(tag='tool')], typing.Annotated[langchain_core.messages.ai.AIMessageChunk, Tag(tag='AIMessageChunk')], typing.Annotated[langchain_core.messages.human.HumanMessageChunk, Tag(tag='HumanMessageChunk')], typing.Annotated[langchain_core.messages.chat.ChatMessageChunk, Tag(tag='ChatMessageChunk')], typing.Annotated[langchain_core.messages.system.SystemMessageChunk, Tag(tag='SystemMessageChunk')], typing.Annotated[langchain_core.messages.function.FunctionMessageChunk, Tag(tag='FunctionMessageChunk')], typing.Annotated[langchain_core.messages.tool.ToolMessageChunk, Tag(tag='ToolMessageChunk')]], FieldInfo(annotation=NoneType, required=True, discriminator=Discriminator(discriminator=<function _get_type at 0x000001A4719A2C00>, custom_error_type=None, custom_error_message=None, custom_error_context=None))]], 'agent_scratchpad': list[typing.Annotated[typing.Union[typing.Annotated[langchain_core.messages.ai.AIMessage, Tag(tag='ai')], typing.Annotated[langchain_core.messages.human.HumanMessage, Tag(tag='human')], typing.Annotated[langchain_core.messages.chat.ChatMessage, Tag(tag='chat')], typing.Annotated[langchain_core.messages.system.SystemMessage, Tag(tag='system')], typing.Annotated[langchain_core.messages.function.FunctionMessage, Tag(tag='function')], typing.Annotated[langchain_core.messages.tool.ToolMessage, Tag(tag='tool')], typing.Annotated[langchain_core.messages.ai.AIMessageChunk, Tag(tag='AIMessageChunk')], typing.Annotated[langchain_core.messages.human.HumanMessageChunk, Tag(tag='HumanMessageChunk')], typing.Annotated[langchain_core.messages.chat.ChatMessageChunk, Tag(tag='ChatMessageChunk')], typing.Annotated[langchain_core.messages.system.SystemMessageChunk, Tag(tag='SystemMessageChunk')], typing.Annotated[langchain_core.messages.function.FunctionMessageChunk, Tag(tag='FunctionMessageChunk')], typing.Annotated[langchain_core.messages.tool.ToolMessageChunk, Tag(tag='ToolMessageChunk')]], FieldInfo(annotation=NoneType, required=True, discriminator=Discriminator(discriminator=<function _get_type at 0x000001A4719A2C00>, custom_error_type=None, custom_error_message=None, custom_error_context=None))]]}, partial_variables={'chat_history': []}, metadata={'lc_hub_owner': 'hwchase17', 'lc_hub_repo': 'openai-functions-agent', 'lc_hub_commit_hash': 'a1655024b06afbd95d17449f21316291e0726f13dcfaf990cc0d18087ad689a5'}, messages=[SystemMessagePromptTemplate(prompt=PromptTemplate(input_variables=[], input_types={}, partial_variables={}, template='You are a helpful assistant'), additional_kwargs={}), MessagesPlaceholder(variable_name='chat_history', optional=True), HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['input'], input_types={}, partial_variables={}, template='{input}'), additional_kwargs={}), MessagesPlaceholder(variable_name='agent_scratchpad')])
| RunnableBinding(bound=ChatOpenAI(client=<openai.resources.chat.completions.completions.Completions object at 0x000001A49035F890>, async_client=<openai.resources.chat.completions.completions.AsyncCompletions object at 0x000001A49035FBC0>, root_client=<openai.OpenAI object at 0x000001A49035FFB0>, root_async_client=<openai.AsyncOpenAI object at 0x000001A49035F0E0>, model_name='gpt-4-turbo', model_kwargs={}, openai_api_key=SecretStr('**********')), kwargs={'tools': [{'type': 'function', 'function': {'name': 'wikipedia', 'description': 'A wrapper around Wikipedia. Useful for when you need to answer general questions about people, places, companies, facts, historical events, or other subjects. Input should be a search query.', 'parameters': {'properties': {'query': {'description': 'query to look up on wikipedia', 'type': 'string'}}, 'required': ['query'], 'type': 'object'}}}, {'type': 'function', 'function': {'name': 'retriever_work_test', 'description': 'retriever performance test.', 'parameters': {'properties': {'__arg1': {'title': '__arg1', 'type': 'string'}}, 'required': ['__arg1'], 'type': 'object'}}}]}, config={}, config_factories=[])
| OpenAIToolsAgentOutputParser(), input_keys_arg=[], return_keys_arg=[], stream_runnable=True), tools=[WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper(wiki_client=<module 'wikipedia' from 'C:\\Users\\gobin\\anaconda3\\Lib\\site-packages\\wikipedia\\__init__.py'>, top_k_results=3, lang='en', load_all_available_meta=False, doc_content_chars_max=500)), Tool(name='Look at LangChain', description='Pleacse give me informatin about LangChain', args_schema=<class 'langchain_core.tools.retriever.RetrieverInput'>, func=functools.partial(<function _get_relevant_documents at 0x000001A475A607C0>, retriever=VectorStoreRetriever(tags=['FAISS', 'OpenAIEmbeddings'], vectorstore=<langchain_community.vectorstores.faiss.FAISS object at 0x000001A484B28A70>, search_kwargs={}), document_prompt=PromptTemplate(input_variables=['page_content'], input_types={}, partial_variables={}, template='{page_content}'), document_separator='\n\n', response_format='content'), coroutine=functools.partial(<function _aget_relevant_documents at 0x000001A475A61BC0>, retriever=VectorStoreRetriever(tags=['FAISS', 'OpenAIEmbeddings'], vectorstore=<langchain_community.vectorstores.faiss.FAISS object at 0x000001A484B28A70>, search_kwargs={}), document_prompt=PromptTemplate(input_variables=['page_content'], input_types={}, partial_variables={}, template='{page_content}'), document_separator='\n\n', response_format='content'))])
```

In [391…

```python
# Invoke user query.

agent_executor_results.invoke({"input":"What is LangChain"})
```

```
> Entering new AgentExecutor chain...

Invoking: `wikipedia` with `{'query': 'LangChain'}`


Page: LangChain
Summary: LangChain is a software framework that helps facilitate the integration of large language models (LLMs) into applications. As a language model integration framework, LangChain's use-cases largely overlap with those of language models in general, including document analysis and summarization, chatbots, and code analysis.

Page: Model Context Protocol
Summary: The Model Context Protocol (MCP) is an open standard, open-source framework introduced by Anthropic in November 20
Invoking: `retriever_work_test` with `LangChain`


retriever_work_test is not a valid tool, try one of [wikipedia, Look at LangChain].LangChain is a software framework designed to facilitate the integration of large language models (LLMs) into applications. It is primarily used for tasks that overlap with the general use-cases of language models, such as document analysis and summarization, chatbot development, and code analysis.

There was a misunderstanding regarding the use of a tool named "retriever_work_test," and it appears to not be a valid tool. If you have any further questions or need additional information about LangChain or related topics, feel free to ask!

> Finished chain.
```

Out[391…

```
{'input': 'What is LangChain',
 'output': 'LangChain is a software framework designed to facilitate the integration of large language models (LLMs) into applications. It is primarily used for tasks that overlap with the general use-cases of language models, such as document analysis and summarization, chatbot development, and code analysis.\n\nThere was a misunderstanding regarding the use of a tool named "retriever_work_test," and it appears to not be a valid tool. If you have any further questions or need additional information about LangChain or related topics, feel free to ask!'}
```

In [393…

```python
agent_executor_results.invoke({"input":" Tell me about Architecture used in Langchain?"})
```

```
> Entering new AgentExecutor chain...

Invoking: `wikipedia` with `{'query': 'Langchain architecture'}`


Page: Milvus (vector database)
Summary: Milvus is a distributed vector database developed by Zilliz. It is available as both open-source software and a cloud service called Zilliz Cloud.
Milvus is an open-source project under the LF AI & Data Foundation and is distributed under the Apache License 2.0.

Page: Sentence embedding
Summary: In natural language processing, a sentence embedding is a representation of a sentence as a vector of numbers which encodes meaningful semantic information.
StateIt appears that there is no specific information available on "Langchain architecture" from the initial search. Langchain may not be well-documented in widely recognized sources like Wikipedia. It may help to look at more specialized technical resources, or directly at the official documentation or website of Langchain, if available, for detailed information regarding its architecture. If you can provide me with more context or specify certain aspects of Langchain you are interested in, I could attempt a more targeted search or look up related technical terms.

> Finished chain.
```

Out[393…

```
{'input': ' Tell me about Architecture used in Langchain?',
 'output': 'It appears that there is no specific information available on "Langchain architecture" from the initial search. Langchain may not be well-documented in widely recognized sources like Wikipedia. It may help to look at more specialized technical resources, or directly at the official documentation or website of Langchain, if available, for detailed information regarding its architecture. If you can provide me with more context or specify certain aspects of Langchain you are interested in, I could attempt a more targeted search or look up related technical terms.'}
```

In [395…

```python
agent_executor_results.invoke({"input":" Please go forward in detail about Prompt Engineering"})
```

```
> Entering new AgentExecutor chain...

Invoking: `wikipedia` with `{'query': 'Prompt Engineering'}`


Page: Prompt engineering
Summary: Prompt engineering is the process of structuring or crafting an instruction in order to produce better outputs from a generative artificial intelligence (AI) model.
A prompt is natural language text describing the task that an AI should perform. A prompt for a text-to-text language model can be a query, a command, or a longer statement including context, instructions, and conversation history. Prompt engineering may involve phrasing a query, specifying a style, **Prompt Engineering** is a key aspect of working with generative Artificial Intelligence (AI) models, especially those trained on large datasets to understand and generate human-like text, such as GPT-3 or similar AI from OpenAI.

### What is Prompt Engineering?
Prompt engineering involves the process of meticulously crafting inputs (known as prompts) to get the most effective and relevant outputs from an AI model. A prompt is typically a piece of text that tells the AI what task to perform, which can range from answering questions, writing essays, generating code, to creating artistic concepts.

### Importance of Prompt Engineering
- **Precision and Clarity:** Clear and well-designed prompts can drastically improve the relevance and accuracy of the AI's outputs.
- **Reducing Ambiguity:** A well-structured prompt reduces misunderstandings and ambiguity, making it easier for the AI to generate the desired results.
- **Exploiting Model Capabilities:** Since AI models often have underlying knowledge gained from their training data, proper prompt engineering can help to exploit this knowledge effectively.

### Techniques in Prompt Engineering
1. **Clarity and Specificity:** Being clear and specific about what you want from the model. General or vague prompts might lead to irrelevant or generic outputs.

2. **Contextual Details:** Providing relevant context in the prompt helps guide the AI, improving the relevance and specificity of its responses.

3. **Prompt Templates:** Creating templates for similar types of queries can help achieve consistent and effective results, especially in a production environment.

4. **Iterative Refinement:** Similar to software testing, prompt engineering often involves an iterative process where prompts are continually refined based on the quality of outputs and new requirements.

5. **Chain of Thought Prompting:** Encouraging the model to "think out loud" by structuring prompts to simulate reasoning steps can often yield more accurate answers, especially in complex problem-solving scenarios.

6. **Zero-shot, Few-shot, and Many-shot Learning:** These techniques involve crafting prompts based on the amount of prior example data given to the model. Zero-shot involves no examples, few-shot includes a few examples, and many-shot utilizes many examples to guide the model's responses.

### Applications of Prompt Engineering
- **Content Creation:** From writing articles to generating creative storytelling and marketing content.
- **Customer Support:** Automating responses in customer service with prompt-driven chatbots.
- **Educational Tools:** Creating tutoring systems that can adapt to the student's learning stage.
- **Programming and Code Generation:** Assisting developers by generating boilerplate code or even complex functions based on descriptive prompts.

### Challenges and Considerations
- **Bias and Ethics:** AI models can inherit biases from training data, so prompts need to be designed to mitigate these biases.
- **Model Limitations:** Understanding the limitations of AI models is crucial as it influences how prompts should be crafted to avoid errors or ethical issues.
- **Security:** Care must be taken to frame prompts in a way that avoids malicious use or unintended information disclosure.

Prompt engineering is thus both an art and a science, requiring a deep understanding of language, technology, and the specific capacities and limitations of AI models. It plays a crucial role in harnessing the full potential of AI technologies in various applications.

> Finished chain.
```

Out[395…

```
{'input': ' Please go forward in detail about Prompt Engineering',
 'output': '**Prompt Engineering** is a key aspect of working with generative Artificial Intelligence (AI) models, especially those trained on large datasets to understand and generate human-like text, such as GPT-3 or similar AI from OpenAI.\n\n### What is Prompt Engineering?\nPrompt engineering involves the process of meticulously crafting inputs (known as prompts) to get the most effective and relevant outputs from an AI model. A prompt is typically a piece of text that tells the AI what task to perform, which can range from answering questions, writing essays, generating code, to creating artistic concepts.\n\n### Importance of Prompt Engineering\n- **Precision and Clarity:** Clear and well-designed prompts can drastically improve the relevance and accuracy of the AI\'s outputs.\n- **Reducing Ambiguity:** A well-structured prompt reduces misunderstandings and ambiguity, making it easier for the AI to generate the desired results.\n- **Exploiting Model Capabilities:** Since AI models often have underlying knowledge gained from their training data, proper prompt engineering can help to exploit this knowledge effectively.\n\n### Techniques in Prompt Engineering\n1. **Clarity and Specificity:** Being clear and specific about what you want from the model. General or vague prompts might lead to irrelevant or generic outputs.\n   \n2. **Contextual Details:** Providing relevant context in the prompt helps guide the AI, improving the relevance and specificity of its responses.\n   \n3. **Prompt Templates:** Creating templates for similar types of queries can help achieve consistent and effective results, especially in a production environment.\n   \n4. **Iterative Refinement:** Similar to software testing, prompt engineering often involves an iterative process where prompts are continually refined based on the quality of outputs and new requirements.\n   \n5. **Chain of Thought Prompting:** Encouraging the model to "think out loud" by structuring prompts to simulate reasoning steps can often yield more accurate answers, especially in complex problem-solving scenarios.\n\n6. **Zero-shot, Few-shot, and Many-shot Learning:** These techniques involve crafting prompts based on the amount of prior example data given to the model. Zero-shot involves no examples, few-shot includes a few examples, and many-shot utilizes many examples to guide the model's responses.\n\n### Applications of Prompt Engineering\n- **Content Creation:** From writing articles to generating creative storytelling and marketing content.\n- **Customer Support:** Automating responses in customer service with prompt-driven chatbots.\n- **Educational Tools:** Creating tutoring systems that can adapt to the student's learning stage.\n- **Programming and Code Generation:** Assisting developers by generating boilerplate code or even complex functions based on descriptive prompts.\n\n### Challenges and Considerations\n- **Bias and Ethics:** AI models can inherit biases from training data, so prompts need to be designed to mitigate these biases.\n- **Model Limitations:** Understanding the limitations of AI models is crucial as it influences how prompts should be crafted to avoid errors or ethical issues.\n- **Security:** Care must be taken to frame prompts in a way that avoids malicious use or unintended information disclosure.\n\nPrompt engineering is thus both an art and a science, requiring a deep understanding of language, technology, and the specific capacities and limitations of AI models. It plays a crucial role in harnessing the full potential of AI technologies in various applications.'}
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```