Step-By-Step Beginner Guide For Creating A Simple AI Agent Using Langchain (Python). This covers the essentials — models, prompts, tools, and agents — and ends with a minimal working example you can copy/paste.

□ LangChain Basics: Step-by-Step Guide to Building an AI Agent

□ 1. Install Requirements

```
pip install langchain langchain-community langchain-openai openai
```

If you plan to use tools (search, calculator, web browsing, etc.), install them as needed.

□ Step 1: Choose an LLM

LangChain supports many models (OpenAI, Gemini, Llama, etc.).

Example: OpenAI

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
```

□ Step 2: Create a Prompt

Prompts tell the model what role to play and how to behave.

```
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI agent."),
    ("user", "{input}")
])
```

□ Step 3: (Optional) Add Tools

Tools allow your agent to *take actions* rather than just answer text.

Example: Calculator Tool

```
from langchain.tools import tool

@tool
def multiply(a: float, b: float):
    """Multiply two numbers."""
    return a * b

tools = [multiply]
```

Other common tools:

- Search tools (Google/Bing)
- Web browsing tools
- File readers
- Database tools
- Custom Python functions

□ Step 4: Create the Agent

LangChain provides several agent types.
For beginners, **AgentExecutor + create_react_agent** is simplest.

```
from langchain.agents import create_react_agent, AgentExecutor

agent = create_react_agent(
    llm=llm,
    tools=tools,
    prompt=prompt
)

agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

□ Step 5: Run the Agent

```
response = agent_executor.invoke({"input": "What is 8 * 7 using the
calculator tool?"})
print(response["output"])
```

□ Minimal Complete Example (Copy & Paste)

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain.tools import tool
from langchain.agents import create_react_agent, AgentExecutor

# 1. LLM
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

# 2. Prompt
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI agent who can use tools."),
    ("user", "{input}")
])

# 3. Tools
@tool
def multiply(a: float, b: float):
    "Multiply two numbers."
    return a * b

tools = [multiply]

# 4. Agent
agent = create_react_agent(
    llm=llm,
    tools=tools,
    prompt=prompt
)

agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# 5. Run
result = agent_executor.invoke({"input": "Use the tool to solve 12 * 9"})
print("Agent Output:", result["output"])
```

□ What You Learned

✔ How to install LangChain
✔ How to create a model
✔ How to build prompts
✔ How to add tools
✔ How to build an agent
✔ How to execute an agent

Below is the **same complete step-by-step LangChain agent guide**, but using **Google Gemini** instead of OpenAI.
This works with **LangChain ≥ 0.2** and Google's official Gemini models.

□ LangChain + Gemini: Step-by-Step Guide to Create an AI Agent

□ 1. Install Required Packages

```
pip install langchain langchain-google-genai google-generativeai
```

□ Step 1: Initialize Gemini LLM

LangChain uses the `GoogleGenerativeAI` wrapper.

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",   # or gemini-1.5-pro
    temperature=0
)
```

Make sure your environment has:

```
export GOOGLE_API_KEY="your_api_key"
```

□ Step 2: Create a Prompt

```
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI agent using Gemini."),
    ("user", "{input}")
])
```

□ Step 3: Add a Tool (Example: Calculator)

LangChain tool definition is the same regardless of model.

```
from langchain.tools import tool

@tool
def multiply(a: float, b: float):
    """Multiply two numbers."""
    return a * b

tools = [multiply]
```

□ Step 4: Create the Agent (ReAct Agent)

```
from langchain.agents import create_react_agent, AgentExecutor

agent = create_react_agent(
    llm=llm,
    tools=tools,
    prompt=prompt
)

agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    verbose=True
)
```

□ Step 5: Run the Agent

```
result = agent_executor.invoke({
    "input": "Use the calculator tool to solve 35 * 6"
})

print("Agent Output:", result["output"])
```

□ Complete Working Code (Copy & Paste)

```
# ----------------------------------------
# LANGCHAIN AI AGENT USING GEMINI
# ----------------------------------------

from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.prompts import ChatPromptTemplate
from langchain.tools import tool
from langchain.agents import create_react_agent, AgentExecutor

# 1. LLM (Gemini)
llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
```

```
    temperature=0
)

# 2. Prompt
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI agent that uses tools when needed."),
    ("user", "{input}")
])

# 3. Tools
@tool
def multiply(a: float, b: float):
    """Multiply two numbers."""
    return a * b

tools = [multiply]

# 4. Agent
agent = create_react_agent(
    llm=llm,
    tools=tools,
    prompt=prompt
)

agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    verbose=True
)

# 5. Execute
result = agent_executor.invoke({"input": "What is 12 * 11 using the tool?"})
print("Agent Output:", result["output"])
```

□ What You Now Have

✔ A Gemini-powered LangChain agent
✔ A working tool (calculator)
✔ A ReAct-style reasoning agent
✔ Fully reusable template