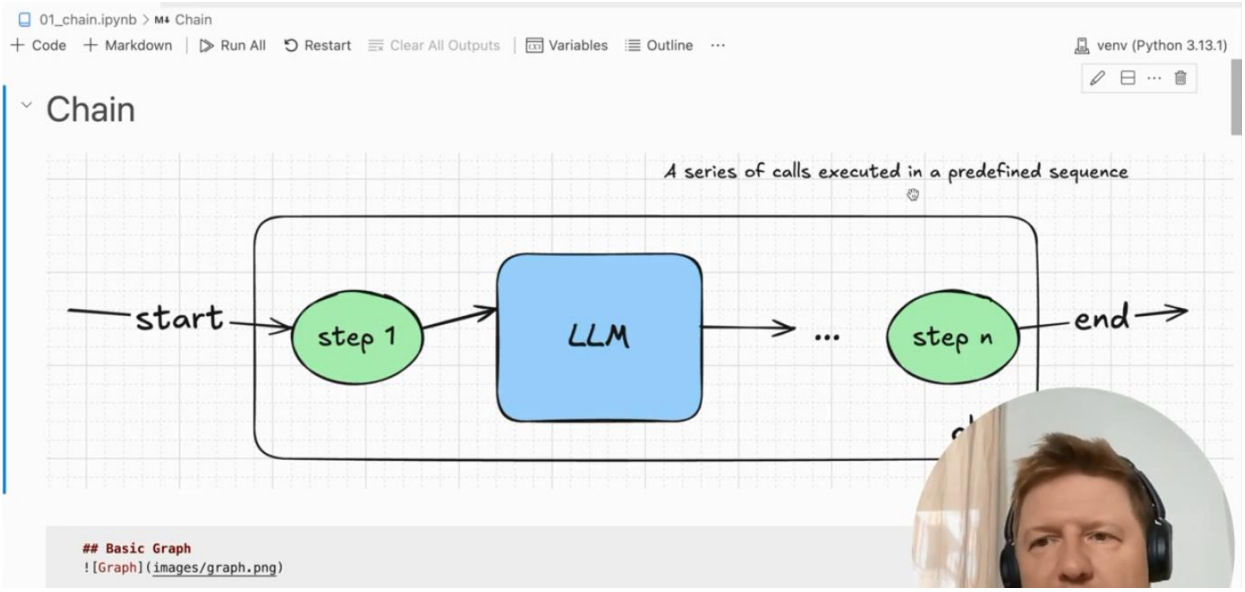
LangGraph Intro - Learn How to Build AI Agents with Tools and Graphs

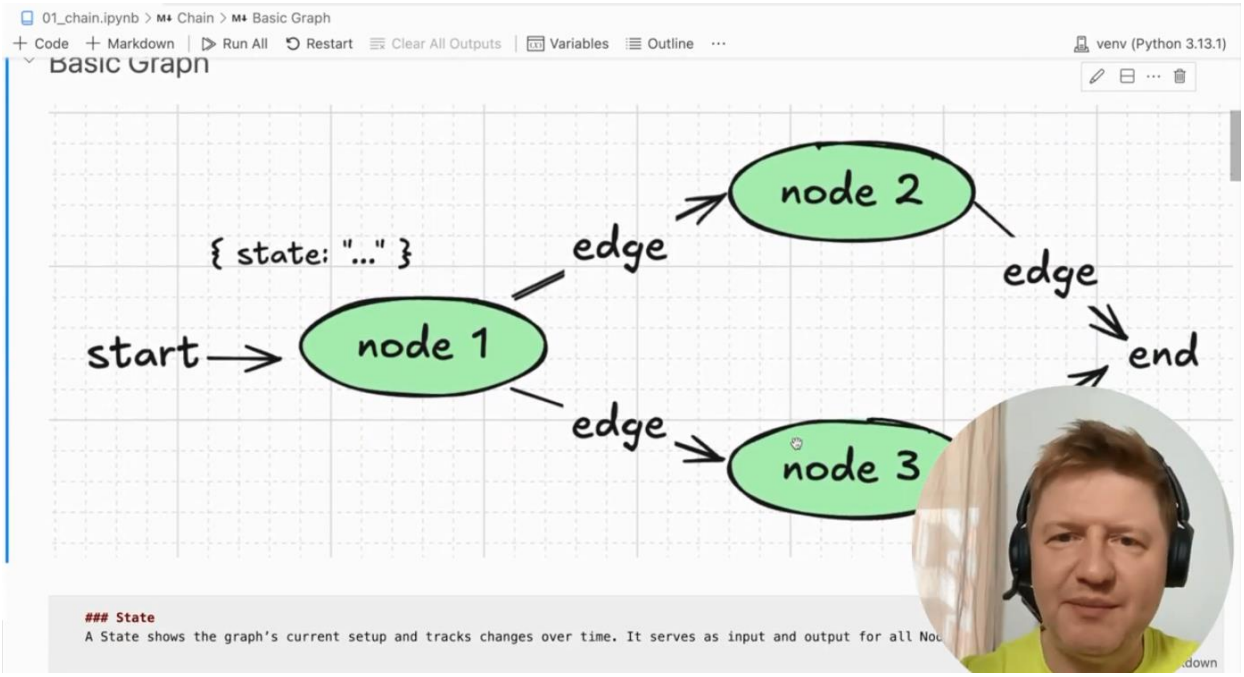
2,215 views Jan 14, 2025

In this video, I'll show you the basics of LangGraph Intro: Learn How to Build AI Agents with Tools and Graphs. If you've been curious about how to build AI agents, how to make AI agents, or even just what LangGraph or LangChain is, this is the perfect place to start!

We'll cover step-by-step how to create simple AI workflows, use tools, and construct graphs in this LangGraph tutorial. It's a beginner-friendly introduction for anyone diving into AI agent tutorials or exploring the Agentic AI Course.

Whether you're new to LangChain agents or looking for a LangChain tutorial, this video makes complex ideas simple. Start your journey into building smarter AI systems today!





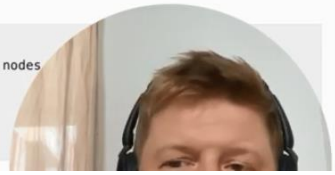
State

A State shows the graph's current setup and tracks changes over time. It serves as input and output for all Nodes and Edges.

```python
from typing_extensions import TypedDict

class State(TypedDict):
    state: str
```

[6] Python

```
### Node
a Node represents an individual element or point within the graph, storing data and connecting with other nodes
relationships.

a Node is a Python function, the first arg of the function is a State
```



Node

a Node represents an individual element or point within the graph, storing data and connecting with other nodes through edges to form relationships.

a Node is a Python function, the first arg of the function is a State

```python
def node_1(state):
    print("Node 1")
    return {"state": state["state"] + "-1-"}

def node_2(state):
    print("Node 2")
    return {"state": state["state"] + "-2-"}

def node_3(state):
    print("Node 3")
    return {"state": state["state"] + "-3-"}
```

[3]

Edges

an Edge represents a connection between two nodes, defining the relationship or interaction between them and potentially carrying additional data. types: - normal edges: always go this way (from node_1 to node_2) - conditional edges: optional route between nodes. a Pythin function that returns a next node based on a logic

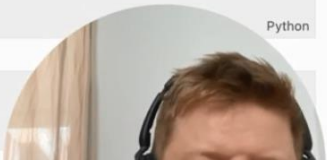
```python
import random
from typing import Literal

def get_random_node(state) -> Literal["node_2", "node_3"]:
    current_state = state['state'] # usually the desiction is based on current state
    if random.random() < 0.5:
        return "node_2"
    return "node_3"
```

[8] Python

+ Code + Markdown

Add Markdown Cell

```
### Graph Construction and Invocation
START and END are special nodes that represent the start and end of the graph.
```

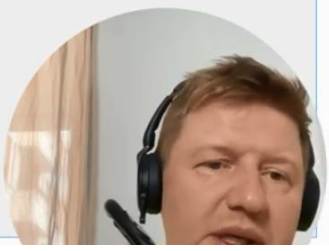


Graph Construction and Invocation

START and END are special nodes that represent the start and end of the graph.

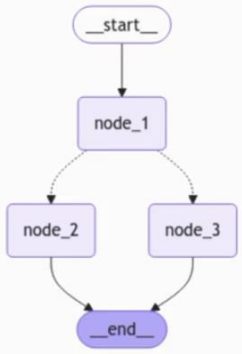
```python
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END

# generate
builder = StateGraph(State)
builder.add_node("node_1", node_1)
builder.add_node("node_2", node_2)
builder.add_node("node_3", node_3)

# logic
builder.add_edge(START, "node_1")
builder.add_conditional_edges("node_1", get_random_node)
builder.add_edge("node_2", END)
builder.add_edge("node_3", END)

# building
graph = builder.compile()

# visualize
display(Image(graph.get_graph().draw_mermaid_png()))
```

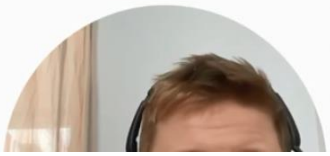
[ ]

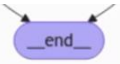


```python
# visualize
display(Image(graph.get_graph().draw_mermaid_png()))
```

[4] ✓ 0.7s Python



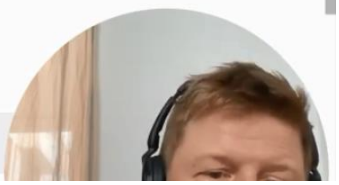





```python
graph.invoke({"state" : "Hi, there!"})
```

[6] ✓ 0.0s Python

```
Node 1
Node 2

{'state': 'Hi, there!-1--2-'}
```



Some LLM related concepts

```python
graph.invoke({"state" : "Hi, there!"})
```

[9] ✓ 0.0s Python

```
Node 1
Node 3

{'state': 'Hi, there!-1--3-'}
```

Some LLM related concepts

Messages

chat models operate on messages. various message types (check out our other Spring AI video):

- HumanMessage
- AIMessage
- SystemMessage
- ToolMessage (will check later)

```python
from pprint import pprint
from langchain_core.messages import AIMessage, HumanMessage, SystemMessage

# Initial SystemMessage to set context
messages = [
    SystemMessage(content="You are a witty and humorous AI assistant for developers, specializing in automating their daily coding headaches."),
]

messages.append(AIMessage(content="Hey there! I'm your AI assistant. Ready to debug your code—or your life?", name="Model"))
messages.append(HumanMessage(content="Can you handle all my TODO comments?", name="Dev"))
messages.append(AIMessage(content="Sure! I'll replace 'TODO: Refactor' with 'TODO: Blame someone else.' Problem solved!", name="Model"))
messages.append(HumanMessage(content="Nice. Can you optimize my SQL too?", name="Dev"))

for message in messages:
    message.pretty_print()
```

[ ]

Chat models

processes messages as prompts and response with completion.

- OpenAI

```
================================ System Message ================================

You are a witty and humorous AI assistant for developers, specializing in automating their daily coding headaches.
================================ Ai Message ================================
Name: Model

Hey there! I'm your AI assistant. Ready to debug your code—or your life?
================================ Human Message ================================
Name: Dev

Can you handle all my TODO comments?
================================ Ai Message ================================
Name: Model

Sure! I'll replace 'TODO: Refactor' with 'TODO: Blame someone else.' Problem solved!
================================ Human Message ================================
Name: Dev
```

Chat models

processes messages as prompts and response with completion.

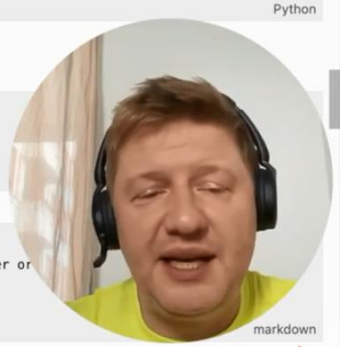
- OpenAI

```python
from langchain_openai import ChatOpenAI

# OPENAI_API_KEY environment variable must be set
llm = ChatOpenAI(model="gpt-4o-mini")
result = llm.invoke(messages)

type(result)
```

Edit ⌘I Chat ⌘L ···

[11] ✓ 1.6s Python

```
langchain_core.messages.ai.AIMessage
```



```python
import json

print(json.dumps(vars(result), indent=4))
```

[ ]

```
### Tools
Tools help an AI use special apps or systems to do things it can't do on its own, like checking the weather or

![Graph](images/tools.png)
```

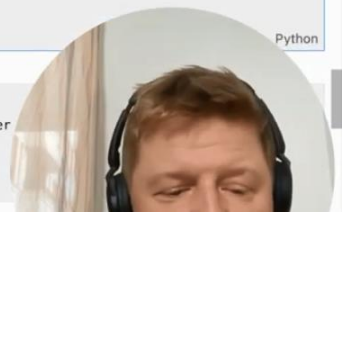
markdown


```python
from langchain_openai import ChatOpenAI

# OPENAI_API_KEY environment variable must be set
llm = ChatOpenAI(model="gpt-4o-mini")
result = llm.invoke(messages)

type(result)
```

Edit ⌘I Chat ⌘L ···

[11] ✓ 1.6s Python

```
langchain_core.messages.ai.AIMessage
```


```python
import json

print(json.dumps(vars(result), indent=4))
```

[ ] Python

```
### Tools
Tools help an AI use special apps or systems to do things it can't do on its own, like checking the weather

![Graph](images/tools.png)
```

01_chain.ipynb > Chain > Some LLM related concepts > Chat models > import json

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· venv (Python 3.13.1)

[11] ✓ 1.6s Python

```
langchain_core.messages.ai.AIMessage
```

```
import json

print(json.dumps(vars(result), indent=4))
```

[12] ✓ 0.0s Python


```
{
    "content": "Absolutely! I'll sprinkle in some SQL fairy dust. Just remember, optimization is like cleaning up after a party\u2014sometimes, you\u20
    "additional_kwargs": {
        "refusal": null
    },
```


01_chain.ipynb > Chain > Some LLM related concepts > Chat models > import json

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· venv (Python 3.13.1)


```
{
    "content": "Absolutely! I'll sprinkle in some SQL fairy dust. Just remember, optimization is like cleaning up after a party\u2014sometimes, you\u20
    "additional_kwargs": {
        "refusal": null
    },
    "response_metadata": {
        "token_usage": {
            "completion_tokens": 52,
            "prompt_tokens": 109,
            "total_tokens": 161,
            "completion_tokens_details": {
                "accepted_prediction_tokens": 0,
                "audio_tokens": 0,
                "reasoning_tokens": 0,
                "rejected_prediction_tokens": 0
            },
            "prompt_tokens_details": {
                "audio_tokens": 0,
                "cached_tokens": 0
            }
        },
        "model_name": "gpt-4o-mini-2024-07-18",
        "system_fingerprint": "fp_bd83329f63",
        "finish_reason": "stop",
        "logprobs": null
...
            "reasoning": 0
        }
    }
}
```

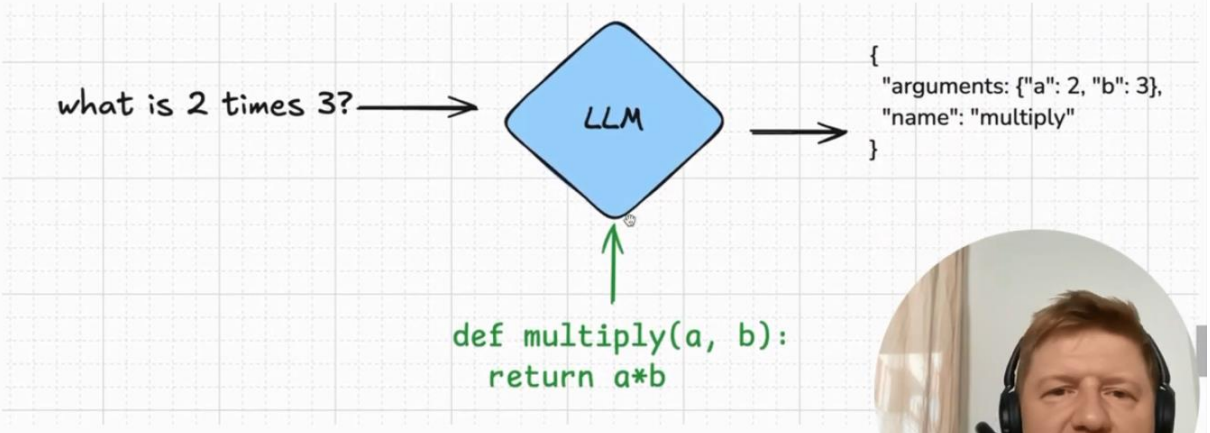

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

01_chain.ipynb > Chain > Some LLM related concepts > Tools

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· venv (Python 3.13.1)

Tools

Tools help an AI use special apps or systems to do things it can't do on its own, like checking the weather or solving tricky problems.




```
def multiply_values(a, b):
    """
    Multiply two values and return the result.
```

Edit ⌘I Chat ⌘L ···

[6]

```python
def multiply_values(a, b):
    """
    Multiply two values and return the result.

    Parameters:
        a (float): The first value.
        b (float): The second value.

    Returns:
        float: The product of a and b.
    """
    return a * b

llm_tools = llm.bind_tools([multiply_values])
```

[6] Python

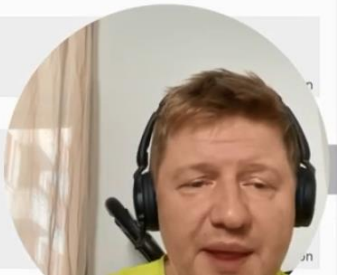
How does LLM know which tool to use?

- the name of the function
- docstring definition
- number of arguments
- ...

```python
tool_call = llm_tools.invoke([HumanMessage(content=f"What is 2 multiplied by 3", name="Dev")])

print(json.dumps(vars(tool_call), indent=4))
```

[ ] Python

+ Code + Markdown



```python
### Merging Messages with State
with LLM we're ineterested in passing messages between nodes. So they become a part of the state.
```

```python
from typing_extensions import TypedDict
from langchain_core.messages import AnyMessage

class MessagesState(TypedDict):
    messages: list[AnyMessage]
```

[10]

```python
tool_call = llm_tools.invoke([HumanMessage(content=f"What is 2 multiplied by 3", name="Dev")])

print(json.dumps(vars(tool_call), indent=4))
```

[14] ✓ 0.9s Python

```
{
    "content": "",
    "additional_kwargs": {
        "tool_calls": [
            {
                "id": "call_NXnTPEoTp5Cz13ysmKNBlKiF",
                "function": {
                    "arguments": "{\"a\":2,\"b\":3}",
                    "name": "multiply_values"
                },
                "type": "function"
            }
        ],
        "refusal": null
    },
    "response_metadata": {
        "token_usage": {
            "completion_tokens": 19,
            "prompt_tokens": 77,
            "total_tokens": 96,
            "completion_tokens_details": {
                "accepted_prediction_tokens": 0,
                "audio_tokens": 0,
                "reasoning_tokens": 0,
                "rejected_prediction_tokens": 0
...
                "reasoning": 0
            }
        }
    }
}
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Merging Messages with State

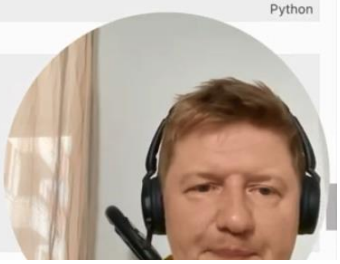
with LLM we're ineterested in passing messages between nodes. So they become a part of the state.

```python
from typing_extensions import TypedDict
from langchain_core.messages import AnyMessage

class MessagesState(TypedDict):
    messages: list[AnyMessage]
```

[10] Python

the problem with this approach - override of the state.
so we need to append messages to the list
we will use reducers for changing the way how state is being updated.

```python
def node_1(state):
    print("Node 1")
    return {"state": state["state"] + "-1-"}
```



```python
from typing import Annotated
from langgraph.graph.message import add_messages

class MessagesState(TypedDict):
    messages: Annotated[list[AnyMessage], add_messages]


initial_messages = [SystemMessage(content="You are a witty and humorous AI assistant for developers, specializing in automating their daily coding
new_message = AIMessage(content="Hey there! I'm your AI assistant. Ready to debug your code—or your life?", name="Model")
add_messages(initial_messages , new_message)
```

[17] ✓ 0.0s Python

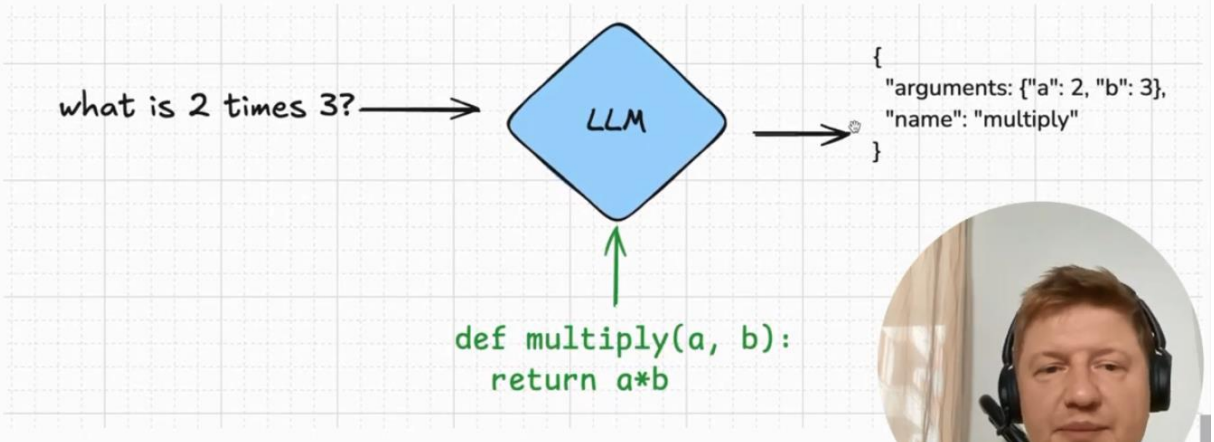
```
[SystemMessage(content='You are a witty and humorous AI assistant for developers, specializing in automating their daily coding ', additional
 AIMessage(content="Hey there! I'm your AI assistant. Ready to debug your code—or your life?", additional_kwargs={}, res ='Model'
```

```python
from langgraph.graph import MessagesState

class MessagesState(MessagesState):
    # Extend to include additional keys beyond the pre-built messages key
    pass
```

[ ]

```markdown
## Combine all together
![Graph](images/tools.png)
```

Combine all together

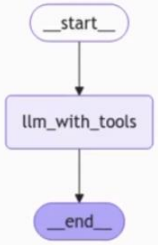



```python
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END
```

```python
from IPython.display import Image, display
from langgraph.graph import StateGraph, START, END

# Node
def llm_with_tools(state: MessagesState):
    return {"messages": [llm_tools.invoke(state["messages"])]}

# Build graph
builder = StateGraph(MessagesState)
builder.add_node("llm_with_tools", llm_with_tools)
builder.add_edge(START, "llm_with_tools")
builder.add_edge("llm_with_tools", END)
graph = builder.compile()

# View
display(Image(graph.get_graph().draw_mermaid_png()))
```

```python
# View
display(Image(graph.get_graph().draw_mermaid_png()))
```

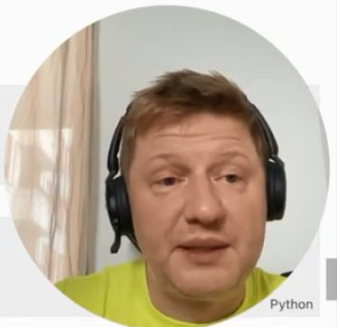
[19] ✓ 0.4s Python




```python
messages = graph.invoke({"messages": HumanMessage(content="How are you?")})
for m in messages['messages']:
    m.pretty_print()
```

```python
messages = graph.invoke({"messages": HumanMessage(content="Multiply 2 and 3")})
for m in messages['messages']:
    m.pretty_print()
```

Python



```python
messages = graph.invoke({"messages": HumanMessage(content="How are you?")})
for m in messages['messages']:
    m.pretty_print()
```

[20] ✓ 8.4s Python

```
================================ Human Message =================================

How are you?
================================== Ai Message ==================================

I'm just a computer program, so I don't have feelings, but I'm here and ready to help you! How can I assist you today?
```

```python
messages = graph.invoke({"messages": HumanMessage(content="Multiply 2 and 3")})
for m in messages['messages']:
    m.pretty_print()
```

[21] ✓ 0.7s Python

```
================================ Human Message =================================

Multiply 2 and 3
================================== Ai Message ==================================
Tool Calls:
  multiply_values (call_Vy5LmId2leiFT73rLpFbxwGO)
 Call ID: call_Vy5LmId2leiFT73rLpFbxwGO
  Args:
    a: 2
    b: 3
```