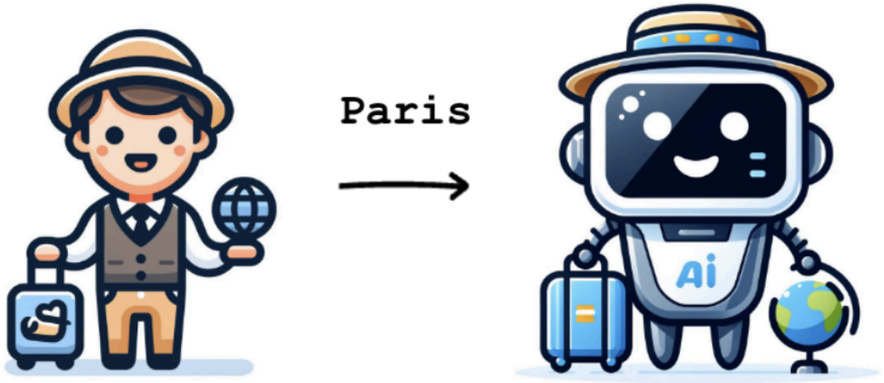
Sequential chains

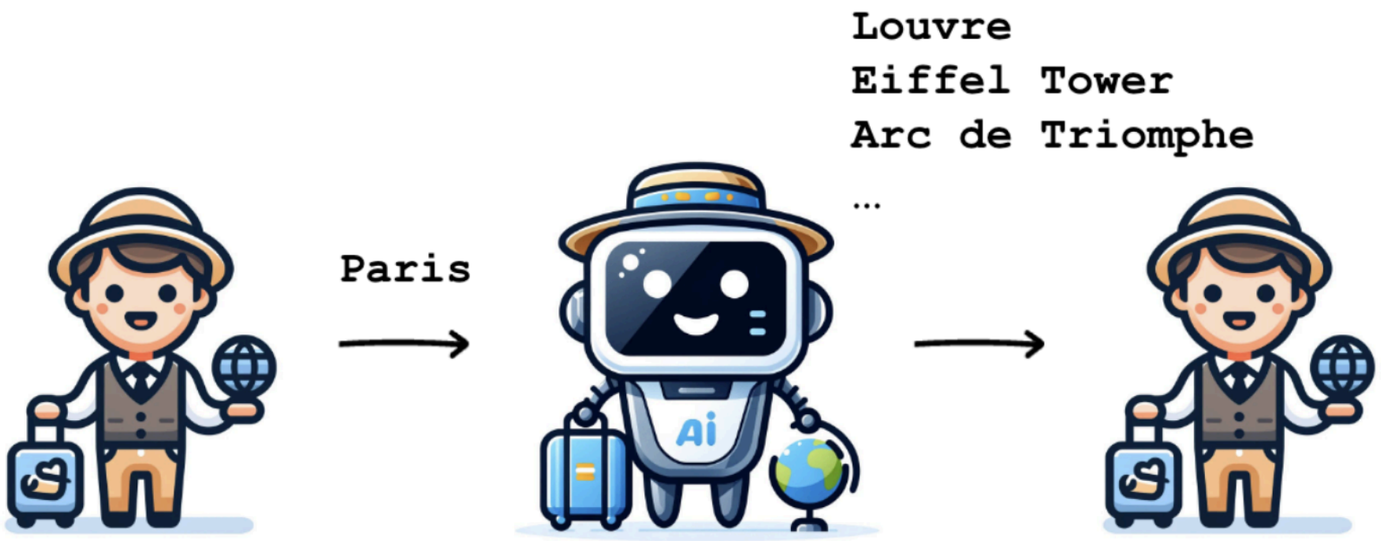
DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

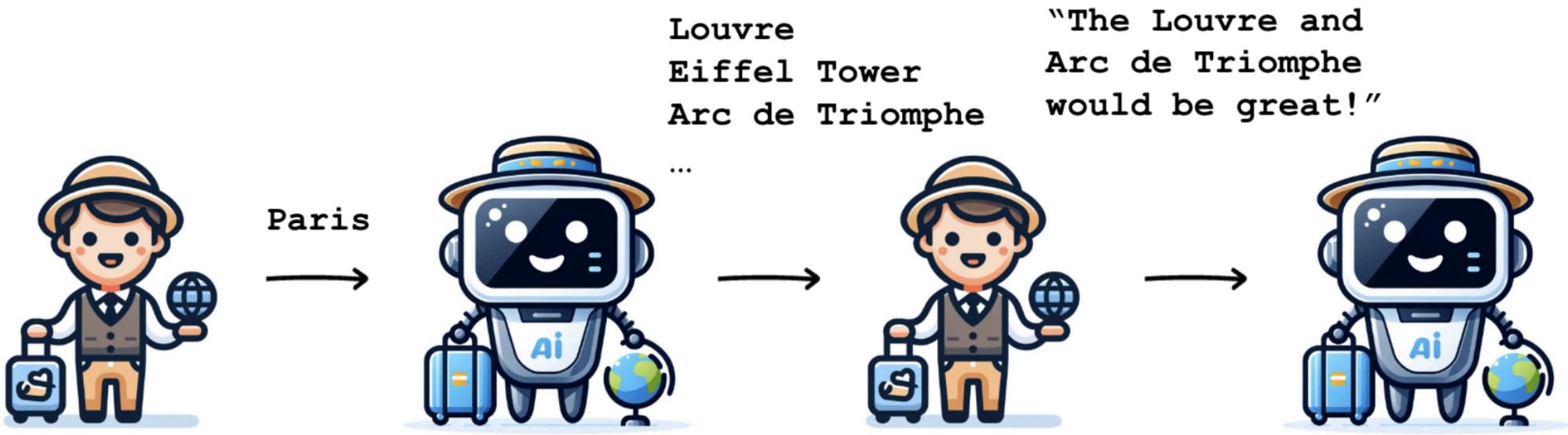
Jonathan Bennion
AI Engineer & LangChain Contributor

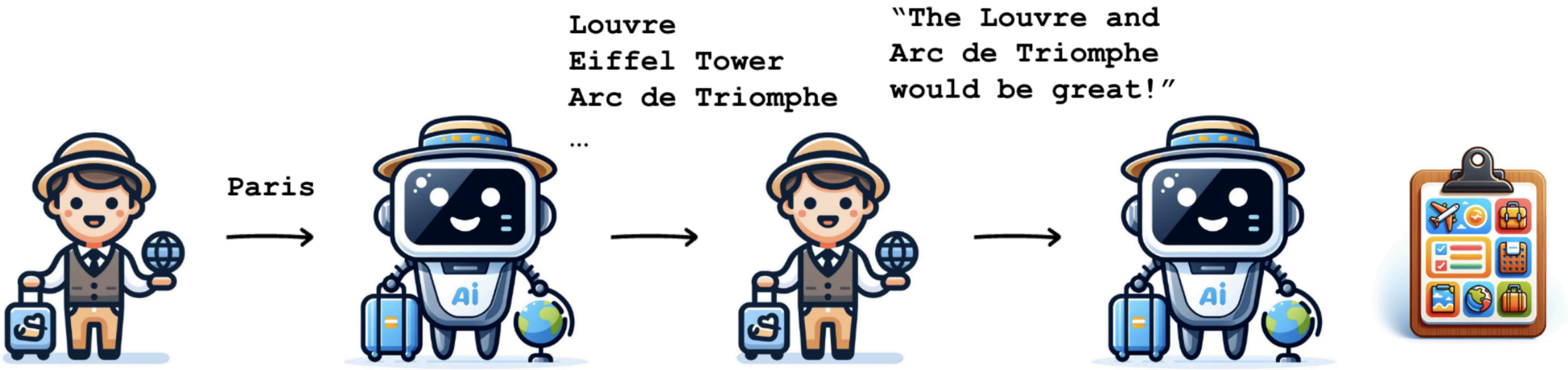
datacamp

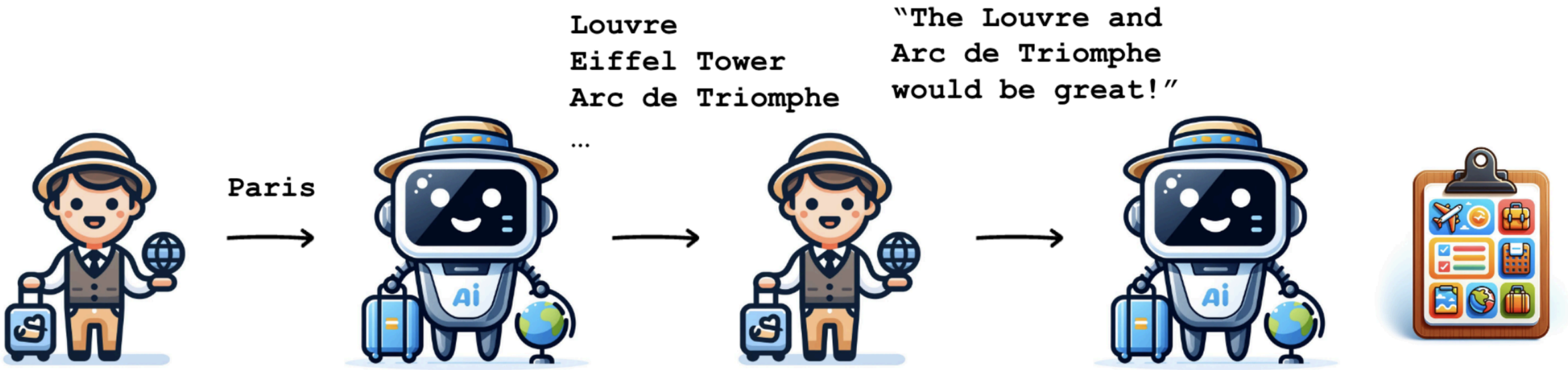

Ai


Paris
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
"The Louvre and
Arc de Triomphe
would be great!"
Ai
Ai


Paris
Louvre
Eiffel Tower
Arc de Triomphe
...
"The Louvre and
Arc de Triomphe
would be great!"
Ai
Ai



SEQUENTIAL PROBLEM

Sequential chains

- Output → input

```
destination_prompt = PromptTemplate(
    input_variables=["destination"],
    template="I am planning a trip to {destination}. Can you suggest some activities to do there?"
)
activities_prompt = PromptTemplate(
    input_variables=["activities"],
    template="I only have one day, so can you create an itinerary from your top three activities: {activities}."
)

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key)

seq_chain = ({"activities": destination_prompt | llm | StrOutputParser()}
    | activities_prompt
    | llm
    | StrOutputParser())
```

```
print(seq_chain.invoke({"destination": "Rome"}))
```

```
- Morning:
1. Start your day early with a visit to the Colosseum. Take a guided tour to learn about its history and significance.
2. After exploring the Colosseum, head to the Roman Forum and Palatine Hill to see more of ancient Rome's ruins.

- Lunch:
3. Enjoy a delicious Italian lunch at a local restaurant near the historic center.

- Afternoon:
4. Visit the Vatican City and explore St. Peter's Basilica, the Vatican Museums, and the Sistine Chapel.
5. Take some time to wander through the charming streets of Rome, stopping at landmarks like the Pantheon, Trevi Fountain, and Piazza Navona.

- Evening:
6. Relax in one of Rome's beautiful parks, such as Villa Borghese or the Orange Garden, for a peaceful escape from the bustling city.
7. End your day with a leisurely dinner at a local restaurant, indulging in more Italian cuisine and maybe some gelato.
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Introduction to LangChain agents

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

datacamp

What are agents?

Agents: use LLMs to take *actions*

Tools: *functions* called by the agent

- **Now** → *ReAct* Agent

ReAct agents

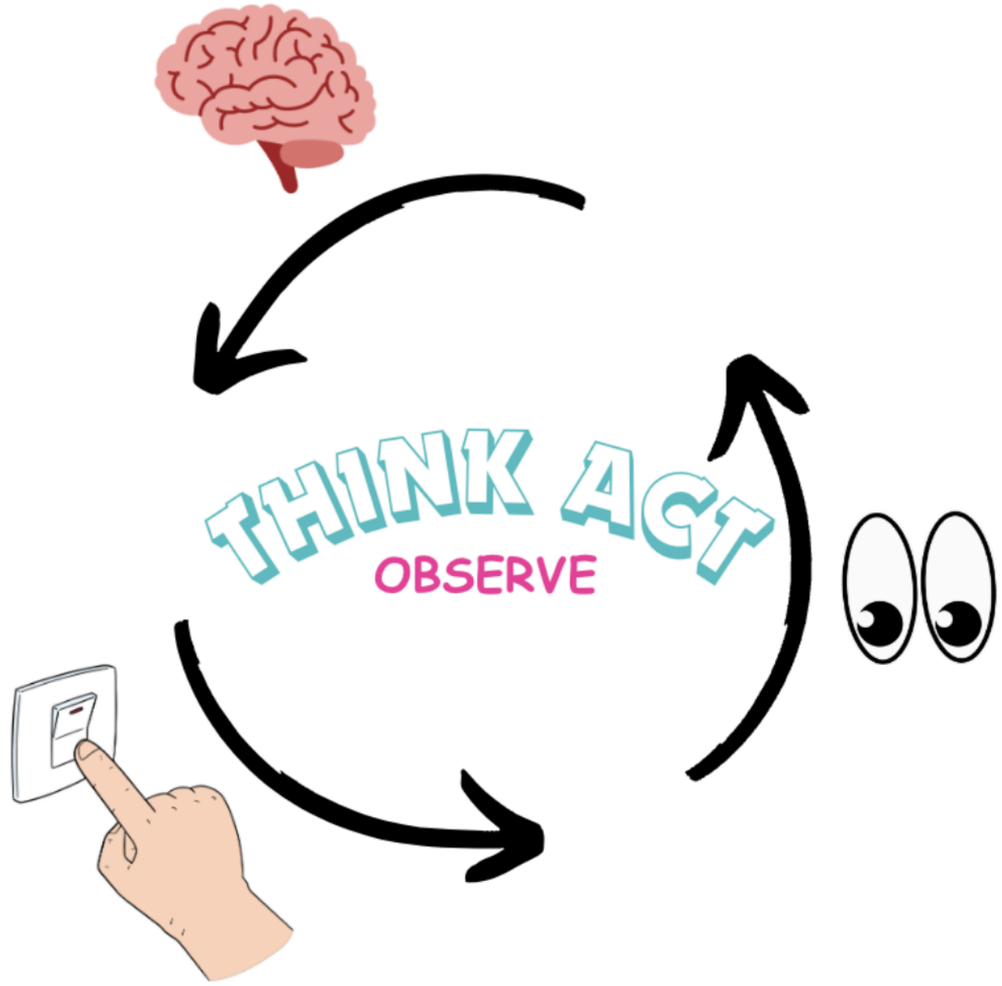
- Reason + Act

What is the weather like in Kingston, Jamaica?

```
Thought: I should call Weather() to find the
weather in Kingston, Jamaica.

Act: Weather("Kingston, Jamaica")

Observe: The weather is mostly sunny with
temperatures of 82°F.
```



LangGraph

- Branch of LangChain centered around designing ***agent systems***
- Unified, tool-agnostic syntax
- `pip install langgraph==0.2.74`

ReAct agent

```
from langgraph.prebuilt import create_react_agent
from langchain_community.agent_toolkits.load_tools import load_tools

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key)
tools = load_tools(["llm-math"], llm=llm)
agent = create_react_agent(llm, tools)

messages = agent.invoke({"messages": [("human", "What is the square root of 101?")]})
print(messages)
```

ReAct agent

```
{'messages': [
    HumanMessage(content='What is the square root of 101?', ...),
    AIMessage(content='', ..., tool_calls=[{'name': 'Calculator', 'args': {'__arg1': 'sqrt(101)'}, ...),
    ToolMessage(content='Answer: 10.049875621120889', ...),
    AIMessage(content='The square root of 101 is approximately 10.05.', ...)
]}
```

```
print(messages['messages'][-1].content)
```

```
The square root of 101 is approximately 10.05.
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Custom tools for agents

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN

Jonathan Bennion
AI Engineer & LangChain Contributor

datacamp

Tool formats

```
from langchain_community.agent_toolkits.load_tools import load_tools

tools = load_tools(["llm-math"], llm=llm)
print(tools[0].name)
```

```
Calculator
```

```
print(tools[0].description)
```

```
Useful for when you need to answer questions about math.
```

- Used by LLM/agent as *context* to determine when to call it

Tool formats

```
print(tools[0].return_direct)
```

```
False
```

Defining a custom function

```
def financial_report(company_name: str, revenue: int, expenses: int) -> str:
    """Generate a financial report for a company that calculates net income."""
    net_income = revenue - expenses

    report = f"Financial Report for {company_name}:\n"
    report += f"Revenue: ${revenue}\n"
    report += f"Expenses: ${expenses}\n"
    report += f"Net Income: ${net_income}\n"
    return report
```

Calling the function

```
print(financial_report(company_name="LemonadeStand", revenue=100, expenses=50))
```

```
Financial Report for LemonadeStand:
Revenue: $100
Expenses: $50
Net Income: $50
```

From functions to tools

```
from langchain_core.tools import tool


@tool
def financial_report(company_name: str, revenue: int, expenses: int) -> str:
    """Generate a financial report for a company that calculates net income."""
    net_income = revenue - expenses

    report = f"Financial Report for {company_name}:\n"
    report += f"Revenue: ${revenue}\n"
    report += f"Expenses: ${expenses}\n"
    report += f"Net Income: ${net_income}\n"
    return report
```

Examining our new tool

```
print(financial_report.name)
print(financial_report.description)
print(financial_report.return_direct)
print(financial_report.args)
```

```
financial_report
Generate a financial report for a company that calculates net income.
False
{'company_name': {'title': 'Company Name', 'type': 'string'},
 'revenue': {'title': 'Revenue', 'type': 'integer'},
 'expenses': {'title': 'Expenses', 'type': 'integer'}}
```

Integrating the custom tool

```
from langgraph.prebuilt import create_react_agent

llm = ChatOpenAI(model="gpt-4o-mini", api_key=openai_api_key, temperature=0)
agent = create_react_agent(llm, [financial_report])

messages = agent.invoke({"messages": [("human", "TechStack generated made $10 millior
with $8 million of costs. Generate a financial report.")]})
print(messages)
```

Integrating the custom tool

```
{'messages': [
    HumanMessage(content='TechStack generated made $10 million dollars with $8 million of...', ...),
    AIMessage(content='', ..., tool_calls=[{'name': 'financial_report',
                                            'args': {'company_name': 'TechStack',
                                                     'revenue': 10000000, 'expenses': 8000000}, ...),
    ToolMessage(content='Financial Report for TechStack:\nRevenue: $10000000\nExpenses...', ...),
    AIMessage(content='Here is the financial report for TechStack...', ...)
]}
```

Tool outputs

```
print(messages['messages'][-1].content)
```

```
Here is the financial report for TechStack:
- Revenue: $10,000,000
- Expenses: $8,000,000
- Net Income: $2,000,000
```

```
Financial Report for TechStack:
Revenue: $10000000
Expenses: $8000000
Net Income: $2000000
```

Let's practice!

DEVELOPING LLM APPLICATIONS WITH LANGCHAIN