Defining Custom Tools

Contents

When constructing your own agent, you will need to provide it with a list of Tools that it can use. Besides the actual function that is called, the Tool consists of several components:

- name (str), is required
- description (str), is optional
- return_direct (bool), defaults to False

The function that should be called when the tool is selected should take as input a single string and return a single string.

There are two ways to define a tool, we will cover both in the example below.

```
# Import things that are needed generically
from langchain.agents import initialize_agent, Tool
from langchain.tools import BaseTool
from langchain.llms import OpenAI
from langchain import LLMMathChain, SerpAPIWrapper
```

Initialize the LLM to use for the agent.

```
llm = OpenAI(temperature=0)
```

Skip to main content

Completely New Tools

First, we show how to create completely new tools from scratch.

There are two ways to do this: either by using the Tool dataclass, or by subclassing the BaseTool class.

Tool dataclass

```
# Load the tool configs that are needed.
search = SerpAPIWrapper()
llm_math_chain = LLMMathChain(llm=llm, verbose=True)
tools = [
    Tool(
        name = "Search",
        func=search.run,
        description="useful for when you need to answer questions about current
events"
    ),
    Tool(
        name="Calculator",
        func=llm_math_chain.run,
        description="useful for when you need to answer questions about math"
    )
]
```

```
# Construct the agent. We will use the default agent type here.
# See documentation for a full list of options.
agent = initialize_agent(tools, llm, agent="zero-shot-react-description",
verbose=True)
```

```
agent.run("Who is Leo DiCaprio's girlfriend? What is her current age raised to the
0.43 power?")
```

```
> Entering new AgentExecutor chain...
 I need to find out who Leo DiCaprio's girlfriend is and then calculate her age
raised to the 0.43 power.
Action: Search
Action Input: "Leo DiCaprio girlfriend"
Observation: Camila Morrone
```

Skip to main content

~~~
Action: Calculator
Action Input: 22^0.43

> Entering new LLMMathChain chain...
22^0.43
```python
import math
print(math.pow(22, 0.43))
```

Answer: 3.777824273683966

> Finished chain.

Observation: Answer: 3.777824273683966

Thought: I now know the final answer
Final Answer: Camila Morrone's age raised to the 0.43 power is 3.777824273683966.

> Finished chain.
~~~

```
"Camila Morrone's age raised to the 0.43 power is 3.777824273683966."
```

Subclassing the BaseTool class

```
class CustomSearchTool(BaseTool):
    name = "Search"
    description = "useful for when you need to answer questions about current
events"

    def _run(self, query: str) -> str:
        """Use the tool."""
        return search.run(query)

    async def _arun(self, query: str) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("BingSearchRun does not support async")

class CustomCalculatorTool(BaseTool):
    name = "Calculator"
    description = "useful for when you need to answer questions about math"

    def _run(self, query: str) -> str:
        """Use the tool."""
        return llm_math_chain.run(query)

    async def _arun(self, query: str) -> str:
```

Skip to main content

```
        """Use the tool asynchronously."""
        raise NotImplementedError("BingSearchRun does not support async")
```

```
tools = [CustomSearchTool(), CustomCalculatorTool()]
```

```
agent = initialize_agent(tools, llm, agent="zero-shot-react-description",
verbose=True)
```

```
agent.run("Who is Leo DiCaprio's girlfriend? What is her current age raised to the
0.43 power?")
```

~~~
> Entering new AgentExecutor chain...
 I need to find out who Leo DiCaprio's girlfriend is and then calculate her age
raised to the 0.43 power.
Action: Search
Action Input: "Leo DiCaprio girlfriend"
Observation: Camila Morrone
Thought: I now need to calculate her age raised to the 0.43 power
Action: Calculator
Action Input: 22^0.43

> Entering new LLMMathChain chain...
22^0.43
```python
import math
print(math.pow(22, 0.43))
```

Answer: 3.777824273683966

> Finished chain.

Observation: Answer: 3.777824273683966

Thought: I now know the final answer
Final Answer: Camila Morrone's age raised to the 0.43 power is 3.777824273683966.

> Finished chain.
~~~

```
"Camila Morrone's age raised to the 0.43 power is 3.777824273683966."
```

Skip to main content

Using the `tool` decorator

To make it easier to define custom tools, a `@tool` decorator is provided. This decorator can be used to quickly create a `Tool` from a simple function. The decorator uses the function name as the tool name by default, but this can be overridden by passing a string as the first argument. Additionally, the decorator will use the function's docstring as the tool's description.

```
from langchain.agents import tool

@tool
def search_api(query: str) -> str:
    """Searches the API for the query."""
    return "Results"
```

```
search_api
```

```
Tool(name='search_api', description='search_api(query: str) -> str - Searches the
API for the query.', return_direct=False, verbose=False, callback_manager=
<langchain.callbacks.shared.SharedCallbackManager object at 0x1184e0cd0>, func=
<function search_api at 0x1635f8700>, coroutine=None)
```

You can also provide arguments like the tool name and whether to return directly.

```
@tool("search", return_direct=True)
def search_api(query: str) -> str:
    """Searches the API for the query."""
    return "Results"
```

```
search_api
```

```
Tool(name='search', description='search(query: str) -> str - Searches the API for
the query.', return_direct=True, verbose=False, callback_manager=
<langchain.callbacks.shared.SharedCallbackManager object at 0x1184e0cd0>, func=
<function search_api at 0x1635f8670>, coroutine=None)
```

Skip to main content

Modify existing tools

Now, we show how to load existing tools and just modify them. In the example below, we do something really simple and change the Search tool to have the name `Google Search`.

```
from langchain.agents import load_tools
```

```
tools = load_tools(["serpapi", "llm-math"], llm=llm)
```

```
tools[0].name = "Google Search"
```

```
agent = initialize_agent(tools, llm, agent="zero-shot-react-description",
verbose=True)
```

```
agent.run("Who is Leo DiCaprio's girlfriend? What is her current age raised to the
0.43 power?")
```

```
> Entering new AgentExecutor chain...
 I need to find out who Leo DiCaprio's girlfriend is and then calculate her age
raised to the 0.43 power.
Action: Google Search
Action Input: "Leo DiCaprio girlfriend"
Observation: Camila Morrone
Thought: I need to find out Camila Morrone's age
Action: Google Search
Action Input: "Camila Morrone age"
Observation: 25 years
Thought: I need to calculate 25 raised to the 0.43 power
Action: Calculator
Action Input: 25^0.43
Observation: Answer: 3.991298452658078

Thought: I now know the final answer
Final Answer: Camila Morrone is Leo DiCaprio's girlfriend and her current age
raised to the 0.43 power is 3.991298452658078.

> Finished chain.
```

Skip to main content

```
"Camila Morrone is Leo DiCaprio's girlfriend and her current age raised to the
0.43 power is 3.991298452658078."
```

Defining the priorities among Tools

When you made a Custom tool, you may want the Agent to use the custom tool more than normal tools.

For example, you made a custom tool, which gets information on music from your database. When a user wants information on songs, You want the Agent to use `the custom tool` more than the normal `Search tool`. But the Agent might prioritize a normal Search tool.

This can be accomplished by adding a statement such as `Use this more than the normal search if the question is about Music, like 'who is the singer of yesterday?' or 'what is the most popular song in 2022?'` to the description.

An example is below.

```
# Import things that are needed generically
from langchain.agents import initialize_agent, Tool
from langchain.llms import OpenAI
from langchain import LLMMathChain, SerpAPIWrapper
search = SerpAPIWrapper()
tools = [
    Tool(
        name = "Search",
        func=search.run,
        description="useful for when you need to answer questions about current
events"
    ),
    Tool(
        name="Music Search",
        func=lambda x: "'All I Want For Christmas Is You' by Mariah Carey.", #Mock
Function
        description="A Music search engine. Use this more than the normal search
if the question is about Music, like 'who is the singer of yesterday?' or 'what is
the most popular song in 2022?'",
    )
]

agent = initialize_agent(tools, OpenAI(temperature=0), agent="zero-shot-react-
description", verbose=True)
```

Skip to main content

```
agent.run("what is the most famous song of christmas")
```

```
> Entering new AgentExecutor chain...
 I should use a music search engine to find the answer
Action: Music Search
Action Input: most famous song of christmas
Observation: 'All I Want For Christmas Is You' by Mariah Carey.
Thought: I now know the final answer
Final Answer: 'All I Want For Christmas Is You' by Mariah Carey.

> Finished chain.
```

```
"'All I Want For Christmas Is You' by Mariah Carey."
```

Using tools to return directly

Often, it can be desirable to have a tool output returned directly to the user, if it's called. You can do this easily with LangChain by setting the return_direct flag for a tool to be True.

```
llm_math_chain = LLMMathChain(llm=llm)
tools = [
    Tool(
        name="Calculator",
        func=llm_math_chain.run,
        description="useful for when you need to answer questions about math",
        return_direct=True
    )
]
```

```
llm = OpenAI(temperature=0)
agent = initialize_agent(tools, llm, agent="zero-shot-react-description",
verbose=True)
```

```
agent.run("whats 2**.12")
```

```
> Entering new AgentExecutor chain...
 I need to calculate this
Action: Calculator
```

```

> Finished chain.
```

```
'Answer: 1.2599210498948732'
```