Defining Custom Tools

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 and must be unique within a set of tools provided to an agent
- description (str), is optional but recommended, as it is used by an agent to determine tool use
- return_direct (bool), defaults to False
- args_schema (Pydantic BaseModel), is optional but recommended, can be used to provide more information (e.g., few-shot examples) or validation for expected parameters.

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

```
# Import things that are needed generically
from langchain import LLMMathChain, SerpAPIWrapper
from langchain.agents import AgentType, initialize_agent
from langchain.chat_models import ChatOpenAI
from langchain.tools import BaseTool, StructuredTool, Tool, tool
```

API Reference:

- AgentType from `langchain.agents`
- initialize_agent from `langchain.agents`
- ChatOpenAI from `langchain.chat_models`
- BaseTool from `langchain.tools`
- StructuredTool from `langchain.tools`
- Tool from `langchain.tools`
- tool from `langchain.tools`

Initialize the LLM to use for the agent.

```
llm = ChatOpenAI(temperature=0)
```

Completely New Tools - String Input and Output

The simplest tools accept a single query string and return a string output. If your tool function requires multiple arguments, you might want to skip down to the `StructuredTool` section below.

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

Tool dataclass

The 'Tool' dataclass wraps functions that accept a single string input and returns a string output.

```
# Load the tool configs that are needed.
search = SerpAPIWrapper()
llm_math_chain = LLMMathChain(llm=llm, verbose=True)
tools = [
    Tool.from_function(
        func=search.run,
        name="Search",
        description="useful for when you need to answer questions about
current events"
        # coroutine= ... <- you can specify an async method if desired as
well
    ),
]
```

```
    /Users/wfh/code/lc/lckg/langchain/chains/llm_math/base.py:50:
UserWarning: Directly instantiating an LLMMathChain with an llm is
deprecated. Please instantiate with llm_chain argument or using the
from_llm class method.
      warnings.warn(
```

You can also define a custom `args_schema`` to provide more information about inputs.

```
from pydantic import BaseModel, Field


class CalculatorInput(BaseModel):
    question: str = Field()
```

```
tools.append(
    Tool.from_function(
        func=llm_math_chain.run,
        name="Calculator",
        description="useful for when you need to answer questions about
math",
        args_schema=CalculatorInput
        # coroutine= ... <- you can specify an async method if desired as
well
    )
)
```

```
# 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=AgentType.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 Leo DiCaprio's girlfriend's name and her age
    Action: Search
    Action Input: "Leo DiCaprio girlfriend"
    Observation: After rumours of a romance with Gigi Hadid, the Oscar
winner has seemingly moved on. First being linked to the television
personality in September 2022, it appears as if his "age bracket" has
moved up. This follows his rumoured relationship with mere 19-year-old
Eden Polani.
    Thought:I still need to find out his current girlfriend's name and age
    Action: Search
    Action Input: "Leo DiCaprio current girlfriend"
    Observation: Just Jared on Instagram: “Leonardo DiCaprio & girlfriend
```

```
Camila Morrone couple up for a lunch date!
    Thought:Now that I know his girlfriend's name is Camila Morrone, I
need to find her current age
    Action: Search
    Action Input: "Camila Morrone age"
    Observation: 25 years
    Thought:Now that I have her age, I need to calculate her age raised to
the 0.43 power
    Action: Calculator
    Action Input: 25^(0.43)

    > Entering new LLMMathChain chain...
    25^(0.43)```text
    25**(0.43)
    ```
    ...numexpr.evaluate("25**(0.43)")...

    Answer: 3.991298452658078
    > Finished chain.

    Observation: Answer: 3.991298452658078
    Thought:I now know the final answer
    Final Answer: Camila Morrone's current age raised to the 0.43 power is
approximately 3.99.

    > Finished chain.

    "Camila Morrone's current age raised to the 0.43 power is
approximately 3.99."
```

## Subclassing the BaseTool class

You can also directly subclass `BaseTool`. This is useful if you want more control over the instance variables or if you want to propagate callbacks to nested chains or other tools.

```
from typing import Optional, Type

from langchain.callbacks.manager import (
```
```

```
    AsyncCallbackManagerForToolRun,
    CallbackManagerForToolRun,
)


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

    def _run(
        self, query: str, run_manager: Optional[CallbackManagerForToolRun]
= None
    ) -> str:
        """Use the tool."""
        return search.run(query)

    async def _arun(
        self, query: str, run_manager:
Optional[AsyncCallbackManagerForToolRun] = None
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("custom_search does not support async")


class CustomCalculatorTool(BaseTool):
    name = "Calculator"
    description = "useful for when you need to answer questions about
math"
    args_schema: Type[BaseModel] = CalculatorInput

    def _run(
        self, query: str, run_manager: Optional[CallbackManagerForToolRun]
= None
    ) -> str:
        """Use the tool."""
        return llm_math_chain.run(query)

    async def _arun(
        self, query: str, run_manager:
Optional[AsyncCallbackManagerForToolRun] = None
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("Calculator does not support async")
```

API Reference:

- AsyncCallbackManagerForToolRun from `langchain.callbacks.manager`
- CallbackManagerForToolRun from `langchain.callbacks.manager`

```
tools = [CustomSearchTool(), CustomCalculatorTool()]
agent = initialize_agent(
    tools, llm, agent=AgentType.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 use custom_search to find out who Leo DiCaprio's girlfriend is, and then use the Calculator to raise her age to the 0.43 power.
    Action: custom_search
    Action Input: "Leo DiCaprio girlfriend"
    Observation: After rumours of a romance with Gigi Hadid, the Oscar winner has seemingly moved on. First being linked to the television personality in September 2022, it appears as if his "age bracket" has moved up. This follows his rumoured relationship with mere 19-year-old Eden Polani.
    Thought:I need to find out the current age of Eden Polani.
    Action: custom_search
    Action Input: "Eden Polani age"
    Observation: 19 years old
    Thought:Now I can use the Calculator to raise her age to the 0.43 power.
    Action: Calculator
    Action Input: 19 ^ 0.43

    > Entering new LLMMathChain chain...
    19 ^ 0.43```text
    19 ** 0.43
    ```
    ...numexpr.evaluate("19 ** 0.43")...

    Answer: 3.547023357958959
    > Finished chain.

    Observation: Answer: 3.547023357958959
    Thought:I now know the final answer.
    Final Answer: 3.547023357958959

    > Finished chain.

    '3.547023357958959'
```

## 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.tools import tool

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

search_api
```

API Reference:
```

- tool from `langchain.tools`

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.', args_schema=<class 'pydantic.main.SearchApi'>,
return_direct=True, verbose=False, callback_manager=
<langchain.callbacks.shared.SharedCallbackManager object at 0x12748c4c0>,
func=<function search_api at 0x16bd66310>, coroutine=None)
```

You can also provide `args_schema` to provide more information about the argument

```
class SearchInput(BaseModel):
    query: str = Field(description="should be a search query")


@tool("search", return_direct=True, args_schema=SearchInput)
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.', args_schema=<class '__main__.SearchInput'>,
return_direct=True, verbose=False, callback_manager=
<langchain.callbacks.shared.SharedCallbackManager object at 0x12748c4c0>,
func=<function search_api at 0x16bcf0ee0>, coroutine=None)
```

Custom Structured Tools

If your functions require more structured arguments, you can use the `StructuredTool` class directly, or still subclass the `BaseTool` class.

StructuredTool dataclass

To dynamically generate a structured tool from a given function, the fastest way to get started is with `StructuredTool.from_function()`.

```
import requests
from langchain.tools import StructuredTool


def post_message(url: str, body: dict, parameters: Optional[dict] = None)
-> str:
    """Sends a POST request to the given url with the given body and
parameters."""
    result = requests.post(url, json=body, params=parameters)
    return f"Status: {result.status_code} - {result.text}"


tool = StructuredTool.from_function(post_message)
```

API Reference:

- StructuredTool from `langchain.tools`

Subclassing the BaseTool

The BaseTool automatically infers the schema from the _run method's signature.

```
from typing import Optional, Type

from langchain.callbacks.manager import (
    AsyncCallbackManagerForToolRun,
    CallbackManagerForToolRun,
)
```

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

    def _run(
        self,
        query: str,
        engine: str = "google",
        gl: str = "us",
        hl: str = "en",
        run_manager: Optional[CallbackManagerForToolRun] = None,
    ) -> str:
        """Use the tool."""
        search_wrapper = SerpAPIWrapper(params={"engine": engine, "gl":
gl, "hl": hl})
        return search_wrapper.run(query)

    async def _arun(
        self,
        query: str,
        engine: str = "google",
        gl: str = "us",
        hl: str = "en",
        run_manager: Optional[AsyncCallbackManagerForToolRun] = None,
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("custom_search does not support async")


# You can provide a custom args schema to add descriptions or custom
validation


class SearchSchema(BaseModel):
    query: str = Field(description="should be a search query")
    engine: str = Field(description="should be a search engine")
    gl: str = Field(description="should be a country code")
    hl: str = Field(description="should be a language code")


class CustomSearchTool(BaseTool):
    name = "custom_search"
    description = "useful for when you need to answer questions about
current events"
    args_schema: Type[SearchSchema] = SearchSchema

    def _run(
        self,
        query: str,
        engine: str = "google",
        gl: str = "us",
        hl: str = "en",
        run_manager: Optional[CallbackManagerForToolRun] = None,
    ) -> str:
        """Use the tool."""
        search_wrapper = SerpAPIWrapper(params={"engine": engine, "gl":
gl, "hl": hl})
        return search_wrapper.run(query)

    async def _arun(
        self,
        query: str,
        engine: str = "google",
        gl: str = "us",
        hl: str = "en",
        run_manager: Optional[AsyncCallbackManagerForToolRun] = None,
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("custom_search does not support async")
```

API Reference:

- AsyncCallbackManagerForToolRun from `langchain.callbacks.manager`
- CallbackManagerForToolRun from `langchain.callbacks.manager`

Using the decorator

The `tool` decorator creates a structured tool automatically if the signature has multiple arguments.

```
import requests
from langchain.tools import tool


@tool
def post_message(url: str, body: dict, parameters: Optional[dict] = None)
-> str:
    """Sends a POST request to the given url with the given body and
parameters."""
    result = requests.post(url, json=body, params=parameters)
    return f"Status: {result.status_code} - {result.text}"
```

API Reference:

- tool from `langchain.tools`

Modify existing tools

Now, we show how to load existing tools and modify them directly. 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
```

API Reference:

- load_tools from `langchain.agents`

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

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

```
agent = initialize_agent(
    tools, llm, agent=AgentType.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 Leo DiCaprio's girlfriend's name and her age.
    Action: Google Search
    Action Input: "Leo DiCaprio girlfriend"
    Observation: After rumours of a romance with Gigi Hadid, the Oscar
winner has seemingly moved on. First being linked to the television
personality in September 2022, it appears as if his "age bracket" has
moved up. This follows his rumoured relationship with mere 19-year-old
Eden Polani.
    Thought:I still need to find out his current girlfriend's name and her
age.
    Action: Google Search
    Action Input: "Leo DiCaprio current girlfriend age"
    Observation: Leonardo DiCaprio has been linked with 19-year-old model
Eden Polani, continuing the rumour that he doesn't date any women over the
age of ...
    Thought:I need to find out the age of Eden Polani.
    Action: Calculator
    Action Input: 19^(0.43)
    Observation: Answer: 3.547023357958959
    Thought:I now know the final answer.
    Final Answer: The age of Leo DiCaprio's girlfriend raised to the 0.43
power is approximately 3.55.

    > Finished chain.





    "The age of Leo DiCaprio's girlfriend raised to the 0.43 power is
approximately 3.55."
```

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.

```python
# Import things that are needed generically
from langchain.agents import initialize_agent, Tool
from langchain.agents import AgentType
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=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True,
)
```

API Reference:

- initialize_agent from `langchain.agents`
- Tool from `langchain.agents`
- AgentType from `langchain.agents`
- OpenAI from `langchain.llms`

```
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'All I Want For Christmas
Is You' by Mariah Carey. 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=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True
)
```

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

```



    > Entering new AgentExecutor chain...
     I need to calculate this
    Action: Calculator
    Action Input: 2**.12Answer: 1.086734862526058

    > Finished chain.





    'Answer: 1.086734862526058'
```

Handling Tool Errors

When a tool encounters an error and the exception is not caught, the agent will stop executing. If you want the agent to continue execution, you can raise a `ToolException` and set `handle_tool_error` accordingly.

When `ToolException` is thrown, the agent will not stop working, but will handle the exception according to the `handle_tool_error` variable of the tool, and the processing result will be returned to the agent as observation, and printed in red.

You can set `handle_tool_error` to `True`, set it a unified string value, or set it as a function. If it's set as a function, the function should take a `ToolException` as a parameter and return a `str` value.

Please note that only raising a `ToolException` won't be effective. You need to first set the `handle_tool_error` of the tool because its default value is `False`.

```python
from langchain.tools.base import ToolException

from langchain import SerpAPIWrapper
from langchain.agents import AgentType, initialize_agent
from langchain.chat_models import ChatOpenAI
from langchain.tools import Tool

from langchain.chat_models import ChatOpenAI


def _handle_error(error: ToolException) -> str:
    return (
        "The following errors occurred during tool execution:"
        + error.args[0]
        + "Please try another tool."
    )


def search_tool1(s: str):
    raise ToolException("The search tool1 is not available.")


def search_tool2(s: str):
    raise ToolException("The search tool2 is not available.")


search_tool3 = SerpAPIWrapper()
```

API Reference:

- ToolException from `langchain.tools.base`
- AgentType from `langchain.agents`
- initialize_agent from `langchain.agents`
- ChatOpenAI from `langchain.chat_models`
- Tool from `langchain.tools`
- ChatOpenAI from `langchain.chat_models`

```
description = "useful for when you need to answer questions about current
events.You should give priority to using it."
tools = [
    Tool.from_function(
        func=search_tool1,
        name="Search_tool1",
        description=description,
        handle_tool_error=True,
    ),
    Tool.from_function(
        func=search_tool2,
        name="Search_tool2",
        description=description,
        handle_tool_error=_handle_error,
    ),
    Tool.from_function(
        func=search_tool3.run,
        name="Search_tool3",
        description="useful for when you need to answer questions about
current events",
    ),
]

agent = initialize_agent(
    tools,
    ChatOpenAI(temperature=0),
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True,
)
```

```
agent.run("Who is Leo DiCaprio's girlfriend?")
```

```
    > Entering new AgentExecutor chain...
    I should use Search_tool1 to find recent news articles about Leo
DiCaprio's personal life.
    Action: Search_tool1
    Action Input: "Leo DiCaprio girlfriend"
    Observation: The search tool1 is not available.
    Thought:I should try using Search_tool2 instead.
    Action: Search_tool2
    Action Input: "Leo DiCaprio girlfriend"
    Observation: The following errors occurred during tool execution:The
search tool2 is not available.Please try another tool.
    Thought:I should try using Search_tool3 as a last resort.
    Action: Search_tool3
    Action Input: "Leo DiCaprio girlfriend"
    Observation: Leonardo DiCaprio and Gigi Hadid were recently spotted at
a pre-Oscars party, sparking interest once again in their rumored romance.
The Revenant actor and the model first made headlines when they were
spotted together at a New York Fashion Week afterparty in September 2022.
    Thought:Based on the information from Search_tool3, it seems that Gigi
Hadid is currently rumored to be Leo DiCaprio's girlfriend.
    Final Answer: Gigi Hadid is currently rumored to be Leo DiCaprio's
girlfriend.

    > Finished chain.




    "Gigi Hadid is currently rumored to be Leo DiCaprio's girlfriend."
```