Tools

Tools are interfaces that an agent can use to interact with the world. They combine a few things:

1. The name of the tool
2. A description of what the tool is
3. JSON schema of what the inputs to the tool are
4. The function to call
5. Whether the result of a tool should be returned directly to the user

It is useful to have all this information because this information can be used to build action-taking systems! The name, description, and JSON schema can be used the prompt the LLM so it knows how to specify what action to take, and then the function to call is equivalent to taking that action.

The simpler the input to a tool is, the easier it is for an LLM to be able to use it. Many agents will only work with tools that have a single string input. For a list of agent types and which ones work with more complicated inputs, please see this documentation

Importantly, the name, description, and JSON schema (if used) are all used in the prompt. Therefore, it is really important that they are clear and describe exactly how the tool should be used. You may need to change the default name, description, or JSON schema if the LLM is not understanding how to use the tool.

Default Tools

Let's take a look at how to work with tools. To do this, we'll work with a built in tool.

```
from langchain_community.tools import
WikipediaQueryRun
from langchain_community.utilities import
WikipediaAPIWrapper
```

Now we initialize the tool. This is where we can configure it as we please

```
api_wrapper =
WikipediaAPIWrapper(top_k_results=1,
doc_content_chars_max=100)
tool =
WikipediaQueryRun(api_wrapper=api_wrapper)
```

This is the default name

```
tool.name
```

```
'Wikipedia'
```

This is the default description

```
tool.description
```

```
'A wrapper around Wikipedia. Useful for when
you need to answer general questions about
people, places, companies, facts, historical
events, or other subjects. Input should be a
search query.'
```

This is the default JSON schema of the inputs

```
tool.args
```

```
{'query': {'title': 'Query', 'type':
'string'}}
```

We can see if the tool should return directly to the user

```
tool.return_direct
```

```
False
```

We can call this tool with a dictionary input

```
tool.run({"query": "langchain"})
```

```
'Page: LangChain\nSummary: LangChain is a framework designed to simplify the creation of applications '
```

We can also call this tool with a single string input. We can do this because this tool expects only a single input. If it required multiple inputs, we would not be able to do that.

```
tool.run("langchain")
```

```
'Page: LangChain\nSummary: LangChain is a framework designed to simplify the creation
```

```
of applications '
```

Customizing Default Tools

We can also modify the built in name, description, and JSON schema of the arguments.

When defining the JSON schema of the arguments, it is important that the inputs remain the same as the function, so you shouldn't change that. But you can define custom descriptions for each input easily.

```
from langchain_core.pydantic_v1 import
BaseModel, Field


class WikiInputs(BaseModel):
    """Inputs to the wikipedia tool."""

    query: str = Field(
        description="query to look up in
Wikipedia, should be 3 or less words"
    )
```

```
tool = WikipediaQueryRun(
    name="wiki-tool",
```

```
    description="look up things in
wikipedia",
    args_schema=WikiInputs,
    api_wrapper=api_wrapper,
    return_direct=True,
)
```

```
tool.name
```

```
'wiki-tool'
```

```
tool.description
```

```
'look up things in wikipedia'
```

```
tool.args
```

```
{'query': {'title': 'Query',
  'description': 'query to look up in
Wikipedia, should be 3 or less words',
  'type': 'string'}}
```

```
tool.return_direct
```

```
True
```

```
tool.run("langchain")
```

```
'Page: LangChain\nSummary: LangChain is a
framework designed to simplify the creation
of applications '
```

More Topics

This was a quick introduction to tools in LangChain, but there is a lot more to learn

Built-In Tools: For a list of all built-in tools, see this page

Custom Tools: Although built-in tools are useful, it's highly likely that you'll have to define your own tools. See this guide for instructions on how to do so.

Toolkits: Toolkits are collections of tools that work well together. For a more in depth description as well as a list of all

built-in toolkits, see this page

Tools as OpenAI Functions: Tools are very similar to OpenAI Functions, and can easily be converted to that format. See this notebook for instructions on how to do that.