Custom multi-action agent

This notebook goes through how to create your own custom agent.

An agent consists of two parts:

```
- Tools: The tools the agent has available to use.
- The agent class itself: this decides which action to take.
```

In this notebook we walk through how to create a custom agent that predicts/takes multiple steps at a time.

```
from langchain.agents import Tool, AgentExecutor, BaseMultiActionAgent
from langchain import OpenAI, SerpAPIWrapper
```

API Reference:

- Tool from `langchain.agents`
- AgentExecutor from `langchain.agents`
- BaseMultiActionAgent from `langchain.agents`

```
def random_word(query: str) -> str:
    print("\nNow I'm doing this!")
    return "foo"
```

```
search = SerpAPIWrapper()
tools = [
    Tool(
        name="Search",
        func=search.run,
        description="useful for when you need to answer questions about
current events",
    ),
    Tool(
        name="RandomWord",
```

```
        func=random_word,
        description="call this to get a random word.",
    ),
]
```

```
from typing import List, Tuple, Any, Union
from langchain.schema import AgentAction, AgentFinish


class FakeAgent(BaseMultiActionAgent):
    """Fake Custom Agent."""

    @property
    def input_keys(self):
        return ["input"]

    def plan(
        self, intermediate_steps: List[Tuple[AgentAction, str]],
**kwargs: Any
    ) -> Union[List[AgentAction], AgentFinish]:
        """Given input, decided what to do.

        Args:
            intermediate_steps: Steps the LLM has taken to date,
                along with observations
            **kwargs: User inputs.

        Returns:
            Action specifying what tool to use.
        """
        if len(intermediate_steps) == 0:
            return [
                AgentAction(tool="Search", tool_input=kwargs["input"],
log=""),
                AgentAction(tool="RandomWord",
tool_input=kwargs["input"], log=""),
            ]
        else:
            return AgentFinish(return_values={"output": "bar"}, log="")

    async def aplan(
        self, intermediate_steps: List[Tuple[AgentAction, str]],
**kwargs: Any
    ) -> Union[List[AgentAction], AgentFinish]:
        """Given input, decided what to do.
```

```
        Args:
            intermediate_steps: Steps the LLM has taken to date,
                along with observations
            **kwargs: User inputs.

        Returns:
            Action specifying what tool to use.
        """
        if len(intermediate_steps) == 0:
            return [
                AgentAction(tool="Search", tool_input=kwargs["input"],
log=""),
                AgentAction(tool="RandomWord",
tool_input=kwargs["input"], log=""),
            ]
        else:
            return AgentFinish(return_values={"output": "bar"}, log="")
```

API Reference:

- AgentAction from `langchain.schema`
- AgentFinish from `langchain.schema`

```
agent = FakeAgent()
```

```
agent_executor = AgentExecutor.from_agent_and_tools(
    agent=agent, tools=tools, verbose=True
)
```

```
agent_executor.run("How many people live in canada as of 2023?")
```

```
    > Entering new AgentExecutor chain...
    The current population of Canada is 38,669,152 as of Monday, April
24, 2023, based on Worldometer elaboration of the latest United Nations
data.
    Now I'm doing this!
    foo

    > Finished chain.
```

```
'bar'
```