Running Agent as an Iterator

It can be useful to run the agent as an interator, to add human-in-the-loop checks as needed.

To demonstrate the `AgentExecutorIterator` functionality, we will set up a problem where an Agent must:

- Retrieve three prime numbers from a Tool
- Multiply these together.

In this simple problem we can demonstrate adding some logic to verify intermediate steps by checking whether their outputs are prime.

```
from langchain.agents import AgentType,
initialize_agent
from langchain.chains import LLMMathChain
from langchain_core.pydantic_v1 import
BaseModel, Field
from langchain_core.tools import Tool
from langchain_openai import ChatOpenAI
```

```
%pip install --upgrade --quiet  numexpr
```

```
# need to use GPT-4 here as GPT-3.5 does not
understand, however hard you insist, that
# it should use the calculator to perform the
final calculation
llm = ChatOpenAI(temperature=0, model="gpt-
4")
llm_math_chain =
LLMMathChain.from_llm(llm=llm, verbose=True)
```

Define tools which provide:

- The `n`th prime number (using a small subset for this example)
- The `LLMMathChain` to act as a calculator

```
primes = {998: 7901, 999: 7907, 1000: 7919}


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


class PrimeInput(BaseModel):
    n: int = Field()
```

```
def is_prime(n: int) -> bool:
    if n <= 1 or (n % 2 == 0 and n > 2):
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True


def get_prime(n: int, primes: dict = primes)
-> str:
    return str(primes.get(int(n)))


async def aget_prime(n: int, primes: dict =
primes) -> str:
    return str(primes.get(int(n)))


tools = [
    Tool(
        name="GetPrime",
        func=get_prime,
        description="A tool that returns the
`n`th prime number",
        args_schema=PrimeInput,
        coroutine=aget_prime,
    ),
    Tool.from_function(
```

```
        func=llm_math_chain.run,
        name="Calculator",
        description="Useful for when you need
to compute mathematical expressions",
        args_schema=CalculatorInput,
        coroutine=llm_math_chain.arun,
    ),
]
```

Construct the agent. We will use OpenAI Functions agent here.

```
from langchain import hub

# Get the prompt to use - you can modify this!
# You can see the full prompt used at:
https://smith.langchain.com/hub/hwchase17/opena
functions-agent
prompt = hub.pull("hwchase17/openai-functions-
agent")
```

```
from langchain.agents import
create_openai_functions_agent

agent = create_openai_functions_agent(llm,
tools, prompt)
```

```
from langchain.agents import AgentExecutor

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

Run the iteration and perform a custom check on certain steps:

```
question = "What is the product of the 998th,
999th and 1000th prime numbers?"

for step in agent_executor.iter({"input":
question}):
    if output :=
step.get("intermediate_step"):
        action, value = output[0]
        if action.tool == "GetPrime":
            print(f"Checking whether {value}
is prime...")
            assert is_prime(int(value))
        # Ask user if they want to continue
        _continue = input("Should the agent
continue (Y/n)?:\n") or "Y"
        if _continue.lower() != "y":
            break
```

```
> Entering new AgentExecutor chain...
```

```
Invoking: `GetPrime` with `{'n': 998}`


7901Checking whether 7901 is prime...
Should the agent continue (Y/n)?:
y

Invoking: `GetPrime` with `{'n': 999}`


7907Checking whether 7907 is prime...
Should the agent continue (Y/n)?:
y

Invoking: `GetPrime` with `{'n': 1000}`


7919Checking whether 7919 is prime...
Should the agent continue (Y/n)?:
y

Invoking: `Calculator` with `{'question':
'7901 * 7907 * 7919'}`



> Entering new LLMMathChain chain...
7901 * 7907 * 7919```text
7901 * 7907 * 7919
```

```
...numexpr.evaluate("7901 * 7907 * 7919")...

Answer: 494725326233
> Finished chain.
Answer: 494725326233Should the agent continue
(Y/n)?:
y
The product of the 998th, 999th and 1000th
prime numbers is 494,725,326,233.

> Finished chain.
```