Stream custom generator functions

You can use generator functions (ie. functions that use the `yield` keyword, and behave like iterators) in a LCEL pipeline.

The signature of these generators should be `Iterator[Input] -> Iterator[Output]`. Or for async generators: `AsyncIterator[Input] -> AsyncIterator[Output]`.

These are useful for: - implementing a custom output parser - modifying the output of a previous step, while preserving streaming capabilities

Let's implement a custom output parser for comma-separated lists.

Sync version

```
%pip install --upgrade --quiet  langchain
langchain-openai
```

```
from typing import Iterator, List

from langchain.prompts.chat import
ChatPromptTemplate
from langchain_core.output_parsers import
StrOutputParser
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_template(
    "Write a comma-separated list of 5
animals similar to: {animal}"
)
model = ChatOpenAI(temperature=0.0)

str_chain = prompt | model |
StrOutputParser()
```

```
for chunk in str_chain.stream({"animal":
"bear"}):
    print(chunk, end="", flush=True)
```

```
lion, tiger, wolf, gorilla, panda
```

```
str_chain.invoke({"animal": "bear"})
```

```
'lion, tiger, wolf, gorilla, panda'
```

```
# This is a custom parser that splits an
iterator of llm tokens
# into a list of strings separated by commas
def split_into_list(input: Iterator[str]) ->
Iterator[List[str]]:
    # hold partial input until we get a comma
    buffer = ""
    for chunk in input:
        # add current chunk to buffer
        buffer += chunk
        # while there are commas in the
buffer
        while "," in buffer:
            # split buffer on comma
            comma_index = buffer.index(",")
            # yield everything before the
comma
            yield
[buffer[:comma_index].strip()]
            # save the rest for the next
iteration
            buffer = buffer[comma_index + 1
:]
```

```
    # yield the last chunk
    yield [buffer.strip()]
```

```
list_chain = str_chain | split_into_list
```

```
for chunk in list_chain.stream({"animal": "bear"}):
    print(chunk, flush=True)
```

```
['lion']
['tiger']
['wolf']
['gorilla']
['panda']
```

```
list_chain.invoke({"animal": "bear"})
```

```
['lion', 'tiger', 'wolf', 'gorilla', 'panda']
```

Async version

```
from typing import AsyncIterator


async def asplit_into_list(
    input: AsyncIterator[str],
) -> AsyncIterator[List[str]]:  # async def
    buffer = ""
    async for (
        chunk
    ) in input:  # `input` is a
`async_generator` object, so use `async for`
        buffer += chunk
        while "," in buffer:
            comma_index = buffer.index(",")
            yield
[buffer[:comma_index].strip()]
            buffer = buffer[comma_index + 1
:]
    yield [buffer.strip()]


list_chain = str_chain | asplit_into_list
```

```
async for chunk in
list_chain.astream({"animal": "bear"}):
    print(chunk, flush=True)
```

```
['lion']
['tiger']
['wolf']
['gorilla']
['panda']
```

```
await list_chain.ainvoke({"animal": "bear"})
```

```
['lion', 'tiger', 'wolf', 'gorilla', 'panda']
```