Run custom functions

You can use arbitrary functions in the pipeline.

Note that all inputs to these functions need to be a SINGLE argument. If you have a function that accepts multiple arguments, you should write a wrapper that accepts a single input and unpacks it into multiple argument.

%pip install –upgrade –quiet langchain langchain-openai

```
from operator import itemgetter

from langchain_core.prompts import
ChatPromptTemplate
from langchain_core.runnables import
RunnableLambda
from langchain_openai import ChatOpenAI


def length_function(text):
    return len(text)
```

```
def _multiple_length_function(text1, text2):
    return len(text1) * len(text2)


def multiple_length_function(_dict):
    return
_multiple_length_function(_dict["text1"],
_dict["text2"])


prompt =
ChatPromptTemplate.from_template("what is {a}
+ {b}")
model = ChatOpenAI()

chain1 = prompt | model

chain = (
    {
        "a": itemgetter("foo") |
RunnableLambda(length_function),
        "b": {"text1": itemgetter("foo"),
"text2": itemgetter("bar")}
        |
RunnableLambda(multiple_length_function),
    }
    | prompt
    | model
)
```

```
chain.invoke({"foo": "bar", "bar": "gah"})
```

```
AIMessage(content='3 + 9 equals 12.')
```

Accepting a Runnable Config

Runnable lambdas can optionally accept a RunnableConfig, which they can use to pass callbacks, tags, and other configuration information to nested runs.

```
from langchain_core.output_parsers import
StrOutputParser
from langchain_core.runnables import
RunnableConfig
```

```
import json


def parse_or_fix(text: str, config:
RunnableConfig):
    fixing_chain = (
        ChatPromptTemplate.from_template(
            "Fix the following
text:\n\n```text\n{input}\n```\nError:
```

```
{error}"
            " Don't narrate, just respond
with the fixed data."
        )
        | ChatOpenAI()
        | StrOutputParser()
    )
    for _ in range(3):
        try:
            return json.loads(text)
        except Exception as e:
            text =
fixing_chain.invoke({"input": text, "error":
e}, config)
    return "Failed to parse"
```

```
from langchain.callbacks import
get_openai_callback

with get_openai_callback() as cb:
    output =
RunnableLambda(parse_or_fix).invoke(
        "{foo: bar}", {"tags": ["my-tag"],
"callbacks": [cb]}
    )
    print(output)
    print(cb)
```

```
{'foo': 'bar'}
Tokens Used: 65
    Prompt Tokens: 56
    Completion Tokens: 9
Successful Requests: 1
Total Cost (USD): $0.00010200000000000001
```