Create a runnable with the `@chain` decorator

You can also turn an arbitrary function into a chain by adding a `@chain` decorator. This is functionaly equivalent to wrapping in a `RunnableLambda`.

This will have the benefit of improved observability by tracing your chain correctly. Any calls to runnables inside this function will be traced as nested childen.

It will also allow you to use this as any other runnable, compose it in chain, etc.

Let's take a look at this in action!

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

```
from langchain_core.output_parsers import
StrOutputParser
from langchain_core.prompts import
```

```
ChatPromptTemplate
from langchain_core.runnables import chain
from langchain_openai import ChatOpenAI
```

```
prompt1 =
ChatPromptTemplate.from_template("Tell me a
joke about {topic}")
prompt2 =
ChatPromptTemplate.from_template("What is the
subject of this joke: {joke}")
```

```
@chain
def custom_chain(text):
    prompt_val1 = prompt1.invoke({"topic":
text})
    output1 =
ChatOpenAI().invoke(prompt_val1)
    parsed_output1 =
StrOutputParser().invoke(output1)
    chain2 = prompt2 | ChatOpenAI() |
StrOutputParser()
    return chain2.invoke({"joke":
parsed_output1})
```

`custom_chain` is now a runnable, meaning you will need to use `invoke`

```
custom_chain.invoke("bears")
```

```
'The subject of this joke is bears.'
```

If you check out your LangSmith traces, you should see a `custom_chain` trace in there, with the calls to OpenAI nested underneath