```
In [ ]:
```

```
In [2]: from langchain_core.prompts import PromptTemplate
        from langchain_openai import ChatOpenAI
        from langchain_core.output_parsers import StrOutputParser  # ✅ Works with y

        # 🔐 Load API keys
        import os
        from dotenv import load_dotenv
        load_dotenv(dotenv_path=".env")
        openai_api_key = os.getenv("OPENAI_API_KEY")

        prompt = PromptTemplate.from_template("Translate to French: {text}")
        llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
        parser = StrOutputParser()

        chain = prompt | llm | parser

        response = chain.invoke({"text": "Good morning"})
        print(response)
```

```
Bonjour
```

```
In [3]: from langchain_core.output_parsers import CommaSeparatedListOutputParser

        prompt = PromptTemplate.from_template("List 5 programming languages, comma-s
        parser = CommaSeparatedListOutputParser()
        chain = prompt | llm | parser

        response = chain.invoke({})
        print(response)  # ['Python', 'Java', 'C++', 'JavaScript', 'Ruby']
```

```
['Python', 'Java', 'C++', 'JavaScript', 'Ruby']
```

In [4]:
```
from langchain_core.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field

class ProductInfo(BaseModel):
    name: str = Field(description="Name of the product")
    price: float = Field(description="Price in INR")

parser = PydanticOutputParser(pydantic_object=ProductInfo)

prompt = PromptTemplate(
    template="Extract product name and price from: {text}\n{format_instructi
    input_variables=["text"],
    partial_variables={"format_instructions": parser.get_format_instructions
)

chain = prompt | llm | parser

response = chain.invoke({
    "text": "The Redmi Note 12 is available for ₹14,999."
})
print(response)
```

```
name='Redmi Note 12' price=14999.0
```

In [7]:
```
from langchain.output_parsers import ResponseSchema, StructuredOutputParser

schemas = [
    ResponseSchema(name="company", description="Name of the company"),
    ResponseSchema(name="founder", description="Name of the founder"),
]

parser = StructuredOutputParser.from_response_schemas(schemas)

prompt = PromptTemplate(
    template="Extract company and founder from the text: {text}\n{format_ins
    input_variables=["text"],
    partial_variables={"format_instructions": parser.get_format_instructions
)

chain = prompt | llm | parser

response = chain.invoke({
    "text": "Hope AI was founded by Ramisha Rani in Tamil Nadu."
})
print(response)
```

```
{'company': 'Hope AI', 'founder': 'Ramisha Rani'}
```

In [ ]:

In [ ]: