Pydantic (JSON) parser

This output parser allows users to specify an arbitrary JSON schema and query LLMs for JSON outputs that conform to that schema.

Keep in mind that large language models are leaky abstractions! You'll have to use an LLM with sufficient capacity to generate well-formed JSON. In the OpenAI family, DaVinci can do reliably but Curie's ability already drops off dramatically.

Use Pydantic to declare your data model. Pydantic's BaseModel like a Python dataclass, but with actual type checking + coercion.

```
from langchain.prompts import (
    PromptTemplate,
    ChatPromptTemplate,
    HumanMessagePromptTemplate,
)
from langchain.llms import OpenAI
from langchain.chat_models import ChatOpenAI
```

API Reference:

- PromptTemplate from `langchain.prompts`
- ChatPromptTemplate from `langchain.prompts`
- HumanMessagePromptTemplate from `langchain.prompts`
- OpenAI from `langchain.llms`
- ChatOpenAI from `langchain.chat_models`

```
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field, validator
from typing import List
```

API Reference:

- PydanticOutputParser from `langchain.output_parsers`

```
model_name = "text-davinci-003"
temperature = 0.0
model = OpenAI(model_name=model_name, temperature=temperature)
```

```
# Define your desired data structure.
class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve the joke")

    # You can add custom validation logic easily with Pydantic.
    @validator("setup")
    def question_ends_with_question_mark(cls, field):
        if field[-1] != "?":
            raise ValueError("Badly formed question!")
        return field


# And a query intented to prompt a language model to populate the data
structure.
joke_query = "Tell me a joke."

# Set up a parser + inject instructions into the prompt template.
parser = PydanticOutputParser(pydantic_object=Joke)

prompt = PromptTemplate(
    template="Answer the user
query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions":
parser.get_format_instructions()},
)

_input = prompt.format_prompt(query=joke_query)

output = model(_input.to_string())

parser.parse(output)
```

```
    Joke(setup='Why did the chicken cross the road?', punchline='To get
to the other side!')
```

```
# Here's another example, but with a compound typed field.
class Actor(BaseModel):
    name: str = Field(description="name of an actor")
    film_names: List[str] = Field(description="list of names of films
they starred in")


actor_query = "Generate the filmography for a random actor."

parser = PydanticOutputParser(pydantic_object=Actor)

prompt = PromptTemplate(
    template="Answer the user
query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions":
parser.get_format_instructions()},
)

_input = prompt.format_prompt(query=actor_query)

output = model(_input.to_string())

parser.parse(output)
```

```
    Actor(name='Tom Hanks', film_names=['Forrest Gump', 'Saving Private
Ryan', 'The Green Mile', 'Cast Away', 'Toy Story'])
```