# OpenAI Function Calling In LangChain

In [2]:
```
import os
import openai

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.environ['OPENAI_API_KEY']
```

In [3]:
```
from typing import List
from pydantic import BaseModel, Field
```

## Pydantic Syntax

Pydantic data classes are a blend of Python's data classes with the validation (making sure of appropriate data types and etc.) power of Pydantic.

They offer a concise way to define data structures while ensuring that the data adheres to specified types and constraints.

In standard python you would create a class like this:

In [4]:
```
class User:
    def __init__(self, name: str, age: int, email: str):
        self.name = name
        self.age = age
        self.email = email
```

In [5]:
```
foo = User(name="Joe",age=32, email="joe@gmail.com")
```

In [6]:
```
foo.name
```

```
'Joe'
```

Lets give the age, a string (and not int)

In [7]:
```
foo = User(name="Joe",age="bar", email="joe@gmail.com")
```

In [8]:
```
foo.age
```

```
'bar'
```

See it just accepted, without any problem.

Now lets make Class with pydantic:

In [9]:
```
class pUser(BaseModel):
    name: str
    age: int
    email: str
```

In [10]:
```
foo_p = pUser(name="Jane", age=32, email="jane@gmail.com")
```

In [11]:
```
foo_p.name
```

```
'Jane'
```

The next cell is expected to fail, because we are giving the inappropriate value to age:

In [12]:
```
foo_p = pUser(name="Jane", age="bar", email="jane@gmail.com")
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[12], line 1
----> 1 foo_p = pUser(name="Jane", age="bar", email="jane@gmail.com")

File /usr/local/lib/python3.9/site-packages/pydantic/main.py:341, in pydanti
c.main.BaseModel.__init__()

ValidationError: 1 validation error for pUser
age
  value is not a valid integer (type=type_error.integer)
```

One more thing of pydantic is that we can nest these data structures:

In [13]:
```
class Class(BaseModel):
    students: List[pUser]
```

In [14]:
```
obj = Class(
    students=[pUser(name="Jane", age=32, email="jane@gmail.com")]
)
```

In [15]:
```
obj
```

```
Class(students=[pUser(name='Jane', age=32, email='jane@gmail.com')])
```

## Pydantic to OpenAI function definition

In [16]:
```
class WeatherSearch(BaseModel):
    #this doc string is important becuase it becomes the description for o
    """Call this with an airport code to get the weather at that airport""

    #this airport_code will become the parameter for openai function
    airport_code: str = Field(description="airport code to get weather for
```

Now we can import "convert_pydantic_to_openai_function" that converts pydantic class to openai function:

In [17]:
```
from langchain.utils.openai_functions import convert_pydantic_to_openai_fun
```

Just convert the above pydantic class to openai function:

In [18]:
```
weather_function = convert_pydantic_to_openai_function(WeatherSearch)
```

In [19]:
```
weather_function
```

```
{'name': 'WeatherSearch',
 'description': 'Call this with an airport code to get the weather at that a
irport',
 'parameters': {'title': 'WeatherSearch',
  'description': 'Call this with an airport code to get the weather at that
airport',
  'type': 'object',
  'properties': {'airport_code': {'title': 'Airport Code',
    'description': 'airport code to get weather for',
    'type': 'string'}},
  'required': ['airport_code']}}
```

As you can see above, we got the openai function.

So if we does not include that description:

In [22]:
```
class WeatherSearch1(BaseModel):
    airport_code: str = Field(description="airport code to get weather for"
```

Note: The next cell is expected to generate an error.

In [23]:
```
convert_pydantic_to_openai_function(WeatherSearch1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[23], line 1
----> 1 convert_pydantic_to_openai_function(WeatherSearch1)

File /usr/local/lib/python3.9/site-packages/langchain/utils/openai_function
s.py:28, in convert_pydantic_to_openai_function(model, name, description)
     24 schema = dereference_refs(model.schema())
     25 schema.pop("definitions", None)
     26 return {
     27     "name": name or schema["title"],
---> 28     "description": description or schema["description"],
     29     "parameters": schema,
     30 }

KeyError: 'description'
```

Desciptions of the parameters are optional in LangChain, If we do not include Description for the Parameter:

In [26]:
```
class WeatherSearch2(BaseModel):
    """Call this with an airport code to get the weather at that airport"""
    airport_code: str
```

In [27]:
```
convert_pydantic_to_openai_function(WeatherSearch2)
```

```
{'name': 'WeatherSearch2',
 'description': 'Call this with an airport code to get the weather at that a
irport',
 'parameters': {'title': 'WeatherSearch2',
  'description': 'Call this with an airport code to get the weather at that
airport',
  'type': 'object',
  'properties': {'airport_code': {'title': 'Airport Code', 'type': 'strin
g'}},
  'required': ['airport_code']}}
```

In [28]:
```
from langchain.chat_models import ChatOpenAI
```

In [35]:
```
model = ChatOpenAI()
```

If we pass that function to model:

In [36]:
```
model.invoke("what is the weather in SF today?", functions=[weather_functio
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "SFO"\n}'}})
```

We can also Bind the function with the model:

```
In [39]: model_with_function = model.bind(functions=[weather_function])
```

```
In [40]: model_with_function.invoke("what is the weather in sf?")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "SFO"\n}'}})
```

## Forcing it to use a function

We can force the model to use a function

```
In [41]: model_with_forced_function = model.bind(functions=[weather_function], funct
```

```
In [42]: model_with_forced_function.invoke("what is the weather in sf?")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "SFO"\n}'}})
```

```
In [43]: model_with_forced_function.invoke("hi!")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "JFK"\n}'}})
```

## Using in a chain

We can use this model bound to function in a chain as we normally would

```
In [44]: from langchain.prompts import ChatPromptTemplate
```

```
In [45]: prompt = ChatPromptTemplate.from_messages([
             ("system", "You are a helpful assistant"),
             ("user", "{input}")
         ])
```

```
In [46]: chain = prompt | model_with_function
```

```
In [47]: chain.invoke({"input": "what is the weather in sf?"})
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "SFO"\n}'}})
```

## Using multiple functions

Even better, we can pass a set of function and let the LLM decide which to use based on the question context.

Introducing another function:

In [48]:
```
class ArtistSearch(BaseModel):
    """Call this to get the names of songs by a particular artist"""
    artist_name: str = Field(description="name of artist to look up")
    n: int = Field(description="number of results")
```

In [49]:
```
functions = [
    convert_pydantic_to_openai_function(WeatherSearch),
    convert_pydantic_to_openai_function(ArtistSearch),
]
```

In [50]:
```
functions
```

```
[{'name': 'WeatherSearch',
  'description': 'Call this with an airport code to get the weather at that
airport',
  'parameters': {'title': 'WeatherSearch',
   'description': 'Call this with an airport code to get the weather at that
airport',
   'type': 'object',
   'properties': {'airport_code': {'title': 'Airport Code',
     'description': 'airport code to get weather for',
     'type': 'string'}},
   'required': ['airport_code']}},
 {'name': 'ArtistSearch',
  'description': 'Call this to get the names of songs by a particular artis
t',
  'parameters': {'title': 'ArtistSearch',
   'description': 'Call this to get the names of songs by a particular artis
t',
   'type': 'object',
   'properties': {'artist_name': {'title': 'Artist Name',
     'description': 'name of artist to look up',
     'type': 'string'},
    'n': {'title': 'N',
     'description': 'number of results',
     'type': 'integer'}},
   'required': ['artist_name', 'n']}}]
```

In [51]:
```
model_with_functions = model.bind(functions=functions)
```

```
In [52]: model_with_functions.invoke("what is the weather in sf?")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'WeatherS
earch', 'arguments': '{\n  "airport_code": "SFO"\n}'}})
```

```
In [53]: model_with_functions.invoke("what are three songs by taylor swift?")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'ArtistSe
arch', 'arguments': '{\n"artist_name": "taylor swift",\n"n": 3\n}'}})
```

```
In [54]: model_with_functions.invoke("hi!")
```

```
AIMessage(content='Hello! How can I assist you today?')
```

```
In [ ]: 
```