Tools and Routing

In [1]:
```
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 [2]:
```
from langchain.agents import tool
```

By putting @tool on above function converts the fucntion to langchain tool, basically toll combines the two main components that are choosing function, parameters and calling the function:

In [4]:
```
@tool
def search(query: str) -> str:
    """Search for weather online"""
    return "42f"
```

So now as it has become a tool:

In [6]:
```
#it has a name
search.name
```

```
'search'
```

In [7]:
```
search.description
```

```
'search(query: str) -> str - Search for weather online'
```

In [8]:
```
search.args
```

```
{'query': {'title': 'Query', 'type': 'string'}}
```

We can use pydantic class to explicitly define the arguements/input schema:

In [9]:
```
from pydantic import BaseModel, Field
class SearchInput(BaseModel):
    query: str = Field(description="Thing to search for")
```

In [13]:
```
#pass argument schema class in arguements
@tool(args_schema=SearchInput)
def search(query: str) -> str:
    """Search for the weather online."""
    return "42f"
```

In [11]:
```
search.args
```

```
{'query': {'title': 'Query',
  'description': 'Thing to search for',
  'type': 'string'}}
```

In [17]:
```
search.run("sf")
```

```
'42f'
```

In [20]:
```
#you can use .invoke too
search.invoke('sf')
```

```
'42f'
```

Finding Temperature using wheather Forecast API:

In [37]:
```
import requests
from pydantic import BaseModel, Field
import datetime

# Define the input schema-->for inputs
class OpenMeteoInput(BaseModel):
    latitude: float = Field(description="Latitude of the location to fetch
    longitude: float = Field(description="Longitude of the location to fetc

#above pydantic class will find out the parameters to use from the input of
#giving input schema in args_schema means that tool will use the above inpu
@tool(args_schema=OpenMeteoInput)
def get_current_temperature(latitude: float, longitude: float) -> dict:
    """Fetch current temperature for given coordinates."""

    #wheather api we will use to get forecasting
    BASE_URL = "https://api.open-meteo.com/v1/forecast"

    # Parameters for the request
    params = {
        'latitude': latitude,
        'longitude': longitude,
        'hourly': 'temperature_2m',
        #one day ahead(next day forecast)
        'forecast_days': 1,
    }

    # Make the request to api
    response = requests.get(BASE_URL, params=params)

    #if we get the response
    if response.status_code == 200:
        results = response.json()
    else:
        raise Exception(f"API Request failed with status code: {response.st

    #current time
    current_utc_time = datetime.datetime.utcnow()

    #finding the time that is close to current time to get the current temp
    time_list = [datetime.datetime.fromisoformat(time_str.replace('Z', '+00
    temperature_list = results['hourly']['temperature_2m']

    closest_time_index = min(range(len(time_list)), key=lambda i: abs(time_
    current_temperature = temperature_list[closest_time_index]

    return f'The current temperature is {current_temperature}°C'
```

In [38]:
```
get_current_temperature.name
```

```
'get_current_temperature'
```

In [39]:
```
get_current_temperature.description
```

```
'get_current_temperature(latitude: float, longitude: float) -> dict - Fetch
current temperature for given coordinates.'
```

In [40]:
```
get_current_temperature.args
```

```
{'latitude': {'title': 'Latitude',
  'description': 'Latitude of the location to fetch weather data for',
  'type': 'number'},
 'longitude': {'title': 'Longitude',
  'description': 'Longitude of the location to fetch weather data for',
  'type': 'number'}}
```

In [41]:
```
from langchain.tools.render import format_tool_to_openai_function
```

Making the tool:

In [42]:
```
format_tool_to_openai_function(get_current_temperature)
```

```
{'name': 'get_current_temperature',
 'description': 'get_current_temperature(latitude: float, longitude: float)
-> dict - Fetch current temperature for given coordinates.',
 'parameters': {'title': 'OpenMeteoInput',
  'type': 'object',
  'properties': {'latitude': {'title': 'Latitude',
    'description': 'Latitude of the location to fetch weather data for',
    'type': 'number'},
   'longitude': {'title': 'Longitude',
    'description': 'Longitude of the location to fetch weather data for',
    'type': 'number'}},
  'required': ['latitude', 'longitude']}}
```

In [35]:
```
get_current_temperature({"latitude": 33.9070, "longitude": 73.3943})
```

```
'The current temperature is 6.8°C'
```

Secondly we want to build a tool thorugh which we can search things on wikipedia:

In [44]:
```
#to get the title of the search from the user input
class get_title(BaseModel):
    title: str= Field(discription= "Title about anything to search about")
```

In [49]:
```
import wikipedia

@tool(args_schema= get_title)
def search_wikipedia(title: str) -> str:
    """Run Wikipedia search and get page summaries according to the title g

    #searching about title on the wikipedia
    page_titles = wikipedia.search(title)

    summaries = []
    #we will consider first three pages
    for page_title in page_titles[: 3]:
        try:
            wiki_page =  wikipedia.page(title=page_title, auto_suggest=Fals
            summaries.append(f"Page: {page_title}\nSummary: {wiki_page.summ
        except (
            self.wiki_client.exceptions.PageError,
            self.wiki_client.exceptions.DisambiguationError,
        ):
            pass
    if not summaries:
        return "No good Wikipedia Search Result was found"
    return "\n\n".join(summaries)
```

In [50]:
```
search_wikipedia.name
```

```
'search_wikipedia'
```

In [51]:
```
search_wikipedia.description
```

```
'search_wikipedia(title: str) -> str - Run Wikipedia search and get page sum
maries according to the title given.'
```

In [52]:
```
format_tool_to_openai_function(search_wikipedia)
```

```
{'name': 'search_wikipedia',
 'description': 'search_wikipedia(title: str) -> str - Run Wikipedia search
and get page summaries according to the title given.',
 'parameters': {'title': 'get_title',
  'type': 'object',
  'properties': {'title': {'title': 'Title',
    'discription': 'Title about anything to search about',
    'type': 'string'}},
  'required': ['title']}}
```

In [58]:
```
search_wikipedia({"title": "details about Machine Learning"})
```

```
'Page: Machine learning in video games\nSummary: In video games, various
artificial intelligence techniques have been used in a variety of ways, r
anging from non-player character (NPC) control to procedural content gene
ration (PCG). Machine learning is a subset of artificial intelligence tha
t focuses on using algorithms and statistical models to make machines act
without specific programming. This is in sharp contrast to traditional me
thods of artificial intelligence such as search trees and expert system
s.\nInformation on machine learning techniques in the field of games is m
ostly known to public through research projects as most gaming companies
choose not to publish specific information about their intellectual prope
rty. The most publicly known application of machine learning in games is
likely the use of deep learning agents that compete with professional hum
an players in complex strategy games. There has been a significant applic
ation of machine learning on games such as Atari/ALE, Doom, Minecraft, St
arCraft, and car racing. Other games that did not originally exists as vi
deo games, such as chess and Go have also been affected by the machine le
arning.\n\nPage: Timeline of machine learning\nSummary: This page is a ti
meline of machine learning. Major discoveries, achievements, milestones a
nd other major events in machine learning are included.\n\n\n\nPage: Boos
```

Sometime functions that we want to use are exposed by the API, and APIs have input and output specification for the access. So we will take these specfications and we will convert them to list of openai function calls:

In [60]:
```
#for spec to openai function
from langchain.chains.openai_functions.openapi import openapi_spec_to_opena
#to load spec
from langchain.utilities.openapi import OpenAPISpec
```

In [61]:
```
text = """
{
  "openapi": "3.0.0",
  "info": {
    "version": "1.0.0",
    "title": "Swagger Petstore",
    "license": {
      "name": "MIT"
    }
  },
  "servers": [
    {
      "url": "http://petstore.swagger.io/v1"
    }
  ],
  "paths": {
    "/pets": {
      "get": {
        "summary": "List all pets",
        "operationId": "listPets",
        "tags": [
          "pets"
        ],
        "parameters": [
          {
            "name": "limit",
            "in": "query",
            "description": "How many items to return at one time (max 100)
            "required": false,
            "schema": {
              "type": "integer",
              "maximum": 100,
              "format": "int32"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "A paged array of pets",
            "headers": {
              "x-next": {
                "description": "A link to the next page of responses",
                "schema": {
                  "type": "string"
                }
              }
            },
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Pets"
                }
              }
            }
          },
          "default": {
            "description": "unexpected error",
```

```
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Error"
              }
            }
          }
        }
      }
    },
    "post": {
      "summary": "Create a pet",
      "operationId": "createPets",
      "tags": [
        "pets"
      ],
      "responses": {
        "201": {
          "description": "Null response"
        },
        "default": {
          "description": "unexpected error",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Error"
              }
            }
          }
        }
      }
    }
  },
  "/pets/{petId}": {
    "get": {
      "summary": "Info for a specific pet",
      "operationId": "showPetById",
      "tags": [
        "pets"
      ],
      "parameters": [
        {
          "name": "petId",
          "in": "path",
          "required": true,
          "description": "The id of the pet to retrieve",
          "schema": {
            "type": "string"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Expected response to a valid request",
          "content": {
            "application/json": {
              "schema": {
```

```
                "$ref": "#/components/schemas/Pet"
              }
            }
          }
        },
        "default": {
          "description": "unexpected error",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Error"
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "Pet": {
        "type": "object",
        "required": [
          "id",
          "name"
        ],
        "properties": {
          "id": {
            "type": "integer",
            "format": "int64"
          },
          "name": {
            "type": "string"
          },
          "tag": {
            "type": "string"
          }
        }
      },
      "Pets": {
        "type": "array",
        "maxItems": 100,
        "items": {
          "$ref": "#/components/schemas/Pet"
        }
      },
      "Error": {
        "type": "object",
        "required": [
          "code",
          "message"
        ],
        "properties": {
          "code": {
            "type": "integer",
            "format": "int32"
```

```
            },
            "message": {
              "type": "string"
            }
          }
        }
      }
    }
}
```

Loading Spec from text:

In [62]:
```
spec = OpenAPISpec.from_text(text)
```

```
Attempting to load an OpenAPI 3.0.0 spec.  This may result in degraded perfo
rmance. Convert your OpenAPI spec to 3.1.* spec for better support.
```

Converting Specs (those input and outputs specifications of api) to openai function, and callables end points of api:

In [63]:
```
pet_openai_functions, pet_callables = openapi_spec_to_openai_fn(spec)
```

In [64]:
```
pet_openai_functions
```

```
[{'name': 'listPets',
  'description': 'List all pets',
  'parameters': {'type': 'object',
   'properties': {'params': {'type': 'object',
     'properties': {'limit': {'type': 'integer',
       'maximum': 100.0,
       'schema_format': 'int32',
       'description': 'How many items to return at one time (max 100)'}},
     'required': []}}}},
 {'name': 'createPets',
  'description': 'Create a pet',
  'parameters': {'type': 'object', 'properties': {}}},
 {'name': 'showPetById',
  'description': 'Info for a specific pet',
  'parameters': {'type': 'object',
   'properties': {'path_params': {'type': 'object',
     'properties': {'petId': {'type': 'string',
       'description': 'The id of the pet to retrieve'}},
     'required': ['petId']}}}}]
```

In [66]:
```
from langchain.chat_models import ChatOpenAI
```

In [67]:
```
model = ChatOpenAI(temperature=0).bind(functions=pet_openai_functions)
```

In [68]:
```
model.invoke("what are three pets names")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'listPet
s', 'arguments': '{\n  "params": {\n    "limit": 3\n  }\n}'}})
```

So we asked it about pets names above so it is telling us to use end point named listpets with limit 3.

In [69]:
```
model.invoke("tell me about pet with id 42")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'showPetB
yId', 'arguments': '{\n  "path_params": {\n    "petId": "42"\n  }\n}'}})
```

We asked it about pet by pet id and so it is telling us to use showPetById.

Routing

In lesson 3, we show an example of function calling deciding between two candidate functions.

Given our tools above, let's format these as OpenAI functions and show this same behavior.

In [70]:
```
#we will convert the tools that we made above to openai function so that we
functions = [
    format_tool_to_openai_function(f) for f in [
        search_wikipedia, get_current_temperature
    ]
]
model = ChatOpenAI(temperature=0).bind(functions=functions)
```

In [73]:
```
model.invoke("what is the weather in Peshawar right now")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'get_curr
ent_temperature', 'arguments': '{\n  "latitude": 34.0150,\n  "longitude": 7
1.5250\n}'}})
```

In [74]:
```
model.invoke("what is Machine Learning")
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'search_w
ikipedia', 'arguments': '{\n  "title": "Machine Learning"\n}'}})
```

In [75]:
```
from langchain.prompts import ChatPromptTemplate
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are helpful but sassy assistant"),
    ("user", "{input}"),
])
chain = prompt | model
```

In [77]:
```
chain.invoke({"input": "what is the weather in Peshawar right now"})
```

```
AIMessage(content='', additional_kwargs={'function_call': {'name': 'get_curr
ent_temperature', 'arguments': '{\n  "latitude": 34.0150,\n  "longitude": 7
1.5249\n}'}})
```

In [78]:
```
from langchain.agents.output_parsers import OpenAIFunctionsAgentOutputParse
```

In [79]:
```
chain = prompt | model | OpenAIFunctionsAgentOutputParser()
```

In [81]:
```
result = chain.invoke({"input": "what is the weather in Peshawar right now"
```

In [82]:
```
type(result)
```

```
langchain.schema.agent.AgentActionMessageLog
```

In [83]:
```
result.tool
```

```
'get_current_temperature'
```

In [84]:
```
result.tool_input
```

```
{'latitude': 34.015, 'longitude': 71.5249}
```

In [85]:
```
get_current_temperature(result.tool_input)
```

```
'The current temperature is 15.6°C'
```

In [86]:
```
result = chain.invoke({"input": "hi!"})
```

In [87]:
```
type(result)
```

```
langchain.schema.agent.AgentFinish
```

In [88]:
```
result.return_values
```

```
{'output': 'Hello! How can I assist you today?'}
```

In [89]:
```
from langchain.schema.agent import AgentFinish
def route(result):
    if isinstance(result, AgentFinish):
        return result.return_values['output']
    else:
        tools = {
            "search_wikipedia": search_wikipedia,
            "get_current_temperature": get_current_temperature,
        }
        return tools[result.tool].run(result.tool_input)
```

In [*]:
```
chain = prompt | model | OpenAIFunctionsAgentOutputParser() | route
```

In [*]:
```
result = chain.invoke({"input": "What is the weather in san francisco right
```

In [*]:
```
result
```

In [ ]:
```
result = chain.invoke({"input": "What is langchain?"})
```

In [ ]:
```
result
```

In [ ]:
```
chain.invoke({"input": "hi!"})
```

In [ ]:
```

```