Function Calling

Function call with Template

pydantic

- Python의 타입 힌트를 실제 데이터 검증과 변환에 사용하는 라이브러리 → 타입 힌트를 실제 작동하는 타입 시스템으로 만들어 주는 도구

① 데이터 검증(Validation) : 타입 힌트를 실제 런타임 검사에 사용
② 데이터 직렬화 / 역직렬화 : JSON ↔ Python 객체 변환 간편
③ API 입력/출력 모델 정의 : API나 LLM Tool의 입력/출력구조를 명확하게 함
④ 데이터 변환 : 문자열 → 숫자 등 자동 변환
⑤ 개발 생산성 : 문서화, 테스트 용이

pydantic 데이터 검증 예

```
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name: str
    age: int

# 자동으로 타입 검증 수행
u = User(id="123", name="Alice", age="25")
print(u)  # id=123 name='Alice' age=25 → 문자열이 int로 자동 변환됨


# 타입이 맞지 않으면 오류 발생
User(id="not_a_number", name="Bob", age="25") # Validation error 발생
```

Pydantic 모델 → JSON Schema 자동 생성

```
from pydantic import BaseModel, Field

class WeatherInput(BaseModel):
    city: str = Field(..., description="조회할 도시 이름")
    unit: str = Field(
        "celsius",
        description="온도 단위",
        json_schema_extra={"enum": ["celsius", "fahrenheit"]}
    )
print(WeatherInput.model_json_schema())
```

{'properties': {'city': {'description': '조회할 도시 이름', 'title': 'City', 'type': 'string'}, 'unit': {'default': 'celsius', 'description': '온도 단위', 'enum': ['celsius', 'fahrenheit'], 'title': 'Unit', 'type': 'string'}}, 'required': ['city'], 'title': 'WeatherInput', 'type': 'object'}

json schema mapping

```
{
  'title': 'WeatherInput',
  'type': 'object',
  'properties': {
    'city': {'title': 'City', 'type': 'string', 'description': '조회할 도시 이름'},
    'unit': {'title': 'Unit', 'type': 'string', 'description': '온도 단위', 'default': 'celsius',
             'enum': ['celsius', 'fahrenheit']}
  },
  'required': ['city']
}
```

① 타입 추출: city:str, unit:str → type:"string"

② 기본값 검사: unit의 기본값 "celsius" 반영

③ 필수 필드 판단: Field(…)는 required, 기본값 있으면 optional # "required": ["city"]

④ Field() 메타데이터 반영: description, enum

⑤ 전체 구조 조합: JSON Schema의 최상위 규격(type=object, properties)으로 감싸기

json schema – 국제 표준

JSON Schema (object)

```
├── title: "모델 이름"
├── type: "object"
├── properties:
│      ├── field1:
│      │      ├── type
│      │      ├── description
│      │      ├── default
│      │      └── enum / constraints
│      └── field2: ...
│
└── required: [필수 필드 목록]
```

Multi-Tool Function Calling with Template

- LangChain과 같은 Framework는 개발자가 Tool 만들고 관리하기 쉽게 라이브러리 제공
- 이 강의에서는 LangChain과 같은 기업용은 아니고, 소규모 에이전트 프로젝트에 적합한 Multi-Tool Function Calling 방법 익힌다.

① Tool 정의 – 개별 Tool 정의하고 구현
② Tool 등록 – 정의된 Tool을 등록하여 Agent가 사용할 수 있도록 함
③ LLM 에이전트 – LLM 기반 multi-tool 활용 코드
④ 실행 예시 – LLM 에이전트를 생성하고 실행

Tool 정의

1. 함수의 Input model 정의

```
class GetWeatherInput(BaseModel):
    city: str = Field(…, description="City name, e.g., 'Seoul'")
    unit: str = Field(default="C", description="Temperature unit 'C' or 'F’”)
```

2. 함수 정의

```
def get_weather(input: GetWeatherInput) -> Dict[str, Any]:
```

Tool Spec 정의

```
class ToolSpec(BaseModel):
    name: str
    description: str
    input_model: Any
    handler: Callable[[Any], Dict[str, Any]]


ToolSpec(
        name="get_weather",
        description="Get mock weather for a city. Educational placeholder (no real API).",
        input_model=GetWeatherInput,
        handler=lambda args: get_weather(GetWeatherInput(**args)),
)
```

Tool Spec → OpenAI tools

```
def as_openai_tool_spec(spec: ToolSpec) -> Dict[str, Any]:
    """Return OpenAI tools[] spec for function calling (JSON Schema)."""
    # pydantic 모델의 JSON schema 활용
    schema = spec.input_model.model_json_schema()
    return {
        "type": "function",
        "function": {
            "name": spec.name,
            "description": spec.description,
            "parameters": schema,
        },
    }
```

Tool Registry

- Tool Spec을 사용하여 실제 Agent와 Interaction하는 클래스

① Tool Spec을 Tool 'name'으로 인덱싱하여 등록 `def register_tool(self, spec: ToolSpec)`

② Tool name으로 해당 handler 함수를 호출 `def call(self, name: str, args: Dict[str, Any]) -> Dict[str, Any]:`

③ 등록된 Tool Spec을 OpenAI Tool 스타일로 반환 `def list_openai_tools(self)`

- Helper function

```
def register_default_tools() -> ToolRegistry:
    reg = ToolRegistry()
    for spec in get_default_tool_specs(): #  Tool 정의에서 구현된 모든 ToolSpec 반환
        reg.register_tool(spec)
    return reg
```

LLM 에이전트

- 에이전트 생성

① 먼저, 가능한 toolspec를 register_default_tools() 함수를 이용하여 registry에 등록

② 등록한 registry를 에이전트 생성 시 argument로 주어서 에이전트가 등록된 툴을 사용할 수 있도록 함

```
def make_agent(use_real_llm: bool = False, model: str = "gpt-4o-mini") -> Any:
    reg = register_default_tools()
    if use_real_llm and os.getenv("OPENAI_API_KEY"):
        return RealLLMAgent(reg, model=model, api_key=os.getenv("OPENAI_API_KEY"))
```

LLM 2차 호출을 위한 메시지 작성

```
if msg.tool_calls:
            messages.append({
                "role": "assistant",
                "content": msg.content or "",
                "tool_calls": [
                    {
                        "id": tc.id,
                        "type": "function",
                        "function": {
                            "name": tc.function.name,
                            "arguments": tc.function.arguments or "{}",
                        },
                    }
                    for tc in (msg.tool_calls or [])
                ],
            })
```

- 이전에는 메시지 전체를 1차 메시지에 append하였는데, 그것 보다는 이렇게 하는 것이 더 분명하고 경제적임.

LLM Tool Call 및 Tool Result의 메시지 추가

```
for tc in msg.tool_calls:
    name = tc.function.name
    args = json.loads(tc.function.arguments or "{}")
    result = self.registry.call(name, args) # ToolRegistry.call()
    tool_outputs.append({"id": tc.id, "name": name, "args": args, "result": result})
    # tool result message 추가
    messages.append({
        "role": "tool",
        "tool_call_id": tc.id,
        "name": name,
        "content": json.dumps(result),
    })
```

Run Demo 예시

```
ap = argparse.ArgumentParser()
ap.add_argument("--use-real-llm", action="store_true", help="Use OpenAI-compatible
function calling")
ap.add_argument("--model", type=str, default="gpt-4o-mini")
args = ap.parse_args()

agent = make_agent(use_real_llm=args.use_real_llm, model=args.model)
# Mock: has respond(); Real: has chat()
run_demo(agent)
```

in run_demo(agent)

out = agent.respond("지금 시간 알려줘") if hasattr(agent, "respond") else agent.chat("지금 시간 알려줘")