
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

5. 사람 개입 *(Human-in-the-Loop)*

- **환경설정**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                                # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_A

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_A
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

```
# ========================================
# 경고 메시지 무시
# ========================================
import os
os.environ['TOKENIZERS_PARALLELISM'] = 'false'
```

```
import sys
from pathlib import Path

# 루트 디렉토리를 Python 경로에 추가
root_dir = Path().absolute().parent
sys.path.append(str(root_dir))

print(f"✅ 루트 디렉토리 추가: {root_dir}")
```

- 응답 시간: `0.0s`
- ✅ 루트 디렉토리 추가: `루트/20250727-langchain-note`

```
from langchain_google_genai import ChatGoogleGenerativeAI

# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
    #model="gemini-3-flash-preview",
    model="gemini-2.5-flash",
    temperature=0.7,                            # gemini-3-flash의 경우 temperatur
    )

result=gemini_lc.invoke("대한민국의 수도는?")

# 테스트
print(result.content)
# print(result.text)
```

- `gemini-2.5-flash` 셀 출력 (`12.7s`)

```
대한민국의 수도는 **서울**입니다.
```

- `gemini-3-flash-preview` 셀 출력 *(아직 미사용)*

---

1) 최신 버전 HITL

- 교재 속 `HITL` 은 v.0.x 또는 그 이전 패턴 기반
- **`최신 LangChain v.1.x` 이상 버전으로 작성**
  - HITL 공식 docu

- **`HumanInTheLoopMiddleware`**: 도구 호출을 사람의 승인이 필요한 경우 중단하고 결정하는 구조로 되고 있음
  - `approve`, `edit`, `reject`
- **`iterrupt_on`**: 어떤 도구에 승인이 필요한지 설정 가능한 옵션
- **`checkpointer` 설정 필수** → 실행의 멈춤 및 재개를 위해서 **`thread_id`** 를 지정해야 함
  - `allowed_decisions`, `description_prefix`, `description` 등의 설정 옵션

---

2) 교재처럼 덧셈 예시 만들어보기

- ① 덧셈 과정을 단계별로 보여주기
- ② 사람이 **`y`** 를 입력해야 다음 단계로 진행하도록 만든 예제

- **`⚠️ 다른 점`**
  - `LangChain`의 `HITL`은 **`콘솔 input`** 이 아니라 **`interrupt`** *(중단)* → **`사람 결정`** → **`resume`** *(재개)* 구조
  - 사람의 **`y`** 입력 = **`approve`** 결정으로 매핑해야 함

- **`✅ 목표 동작`**
  - 문제: 12 + 35
  - `Step 1.` 일의 자리 계산: 2 + 5 = 7
  - 👉 **계속할까요? (y)**
  - `Step 2.` 십의 자리 계산: 1 + 3 = 4
  - 👉 **계속할까요? (y)**
  - `최종 결과`: 47

- **`✅ 핵심 아이디어`**

- 각 단계마다 `HITL interrupt`가 발생
- 사람의
  - **`approve`** = **`y`**
  - **`reject`** = **`n`**
- **`checkpointer`** 로 상태 유지

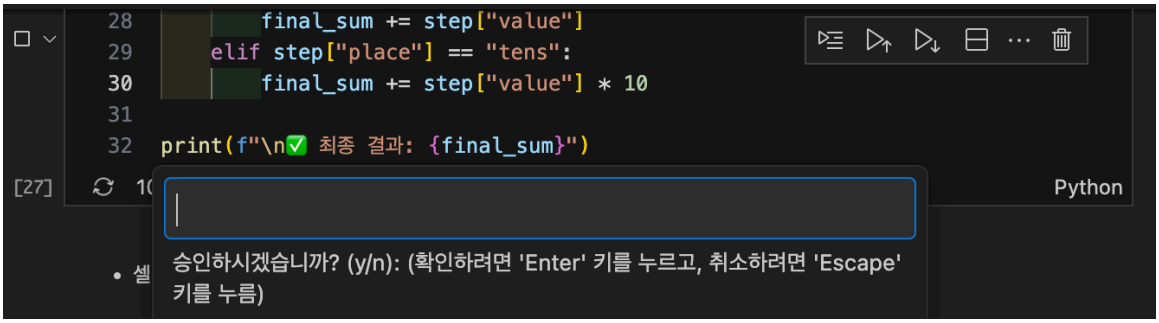
```
# 임포트

from langchain.agents.middleware import HumanInTheLoopMiddleware
from langchain_core.runnables import RunnableLambda
import uuid
```

```
# HITL 미들웨어 설정

hitl = HumanInTheLoopMiddleware(
    interrupt_on={
        "add_step": {
        # step 키 하위에 설정을 딕셔너리로 넣기
            "allowed_decisions" :["approve", "reject"],
            "description" : "이 단계가 맞으면 y(approve)를 눌러 다음 단계로 진행하세요."
        },
    },
    description_prefix="📘 덧셈 풀이 단계",
)
```

```
def add_step_by_step(inputs):
    # sourcery skip: merge-list-append, move-assign-in-block
    a, b = inputs["a"], inputs["b"]

    a_ones, a_tens = a % 10, a // 10
    b_ones, b_tens = b % 10, b // 10

    steps = []

    # Step 1: 일의 자리
    ones_sum = a_ones + b_ones
    steps.append({
        "text": f"Step 1️⃣ 일의 자리 계산: {a_ones} + {b_ones} = {ones_sum}",
        "value": ones_sum,
        "place": "ones"
    })

    # Step 2: 십의 자리
    tens_sum = a_tens + b_tens
    steps.append({
        "text": f"Step 2️⃣ 십의 자리 계산: {a_tens} + {b_tens} = {tens_sum}",
        "value": tens_sum,
        "place": "tens"
    })

    return steps
```

```
# Runnable 체인 구성하기

step_chain = RunnableLambda(add_step_by_step)
```

```
# 실행 + 사람 승인 흐름
from langgraph.types import interrupt
from langgraph.checkpoint.memory import MemorySaver
import uuid

# HITL은 상태 저장이 필수이므로 checkpointer가 필요함
memory = MemorySaver()
thread_id = {"configurable": {"thread_id": str(uuid.uuid4())}}

steps = add_step_by_step({"a": 12, "b": 35})
final_sum = 0

for step in steps:
    print(step["text"])

    # ❌ HumanInTheLoopMiddleware.interrupt(...) 대신
    # ✅ 아래와 같이 'interrupt' 함수를 사용하거나, 에이전트 스트림을 사용해야 함
    # 여기서는 개념 습득을 위해 수동 중단을 흉내내는 코드로 바꾸기

    print(f"👉 [승인 대기 중]: {step['text']}")
    # 실제 에이전트 환경이라면 여기서 제어권이 사용자에게 넘어감
    user_input = input("승인하시겠습니까? (y/n): ")

    if user_input.lower() != 'y':
        print("❌ 사용자가 거절했습니다. 중단합니다.")
        break
    if step["place"] == "ones":
        final_sum += step["value"]
    elif step["place"] == "tens":
        final_sum += step["value"] * 10

print(f"\n✅ 최종 결과: {final_sum}")
```

- 셀 출력 (`4.6s`) - **실제 과정을 보기 위해 수동으로 함수 과정으로 바꿔봄**
  - Step 1️⃣ 일의 자리 계산: 2 + 5 = 7
  - 👉 [승인 대기 중]: Step 1️⃣ 일의 자리 계산: 2 + 5 = 7
    - 

      - *거절한 경우:* ❌ *사용자가 거절했습니다. 중단합니다.*
  - Step 2️⃣ 십의 자리 계산: 1 + 3 = 4

    - 👉 [승인 대기 중]: Step 2️⃣ 십의 자리 계산: 1 + 3 = 4
    - ✅ 최종 결과: 47

- 올바른 방식: `LangGraph` + `interrupt` → `approve` → `resume`

```
# 단계별 덧셈 + HITL
from langgraph.graph import StateGraph
from langgraph.types import interrupt
from typing import TypedDict
from langgraph.checkpoint.memory import MemorySaver
```

```
# 상태 정의

class AddState(TypedDict):
    a: int
    b: int
    step: int
    #result: int
    sum_result: int
```

- 첫번째 시도

```
# Step 1 – 일의 자리 (interrupt 발생)

def ones_step(state: AddState):
    a_ones = state["a"] % 10
    b_ones = state["b"] % 10
    s = a_ones + b_ones

    interrupt(
        f"Step 1️⃣ 일의 자리 계산: {a_ones} + {b_ones} = {s}\n"
        f"맞으면 approve(y)를 누르세요."
    )

    return {
        **state,
        "result": s,
        "step": 2
    }

# Step 2 – 십의 자리

def tens_step(state: AddState):
    a_tens = state["a"] // 10
    b_tens = state["b"] // 10
    s = a_tens + b_tens

    interrupt(
```

```
        f"Step 2️⃣ 십의 자리 계산: {a_tens} + {b_tens} = {s}\n"
        f"맞으면 approve(y)를 누르세요."
    )

    return {
        **state,
        "result": state["result"] + s * 10,
        "step": 3
    }

# 그래프 구성
graph = StateGraph(AddState)

graph.add_node("ones", ones_step)
graph.add_node("tens", tens_step)

graph.set_entry_point("ones")
graph.add_edge("ones", "tens")

app = graph.compile()
```

- 두번째 시도

```
from langgraph.types import interrupt

def add_node(state: AddState):
    a, b, step = state["a"], state["b"], state["step"]

    if step == 1:
        ones = (a % 10) + (b % 10)
        interrupt(f"Step 1️⃣ 일의 자리 계산: {a%10} + {b%10} = {ones}")
        return {
            "step": 2,
            "result": ones
        }

    if step == 2:
        tens = (a // 10) + (b // 10)
        interrupt(f"Step 2️⃣ 십의 자리 계산: {a//10} + {b//10} = {tens}")
        return {
            "step": 3,
            "result": state["result"] + tens * 10
        }

    return state
```

```
# 그래프 구성

graph = StateGraph(AddState)

graph.add_node("ones", ones_step)
graph.add_node("tens", tens_step)

graph.set_entry_point("ones")
graph.add_edge("ones", "tens")

checkpointer = MemorySaver()
app = graph.compile(checkpointer=checkpointer)
```

- 세번째 시도

```
from langgraph.types import interrupt

def add_node(state):
    a = state["a"]
    b = state["b"]
    step = state["step"]

    # 1단계
    if step == 1:
        ones = (a % 10) + (b % 10)
        # ✅ 먼저 상태를 업데이트하고 반환
        new_state = {
            **state,
            "step": 2,
            "sum_result": ones
        }
        # ✅ interrupt는 상태 업데이트 후에 호출
        interrupt(f"Step 1️⃣ 일의 자리 계산: {a%10} + {b%10} = {ones}")
        return new_state

    # 2단계
    if step == 2:
        tens = (a // 10) + (b // 10)
        # ✅ 먼저 상태를 업데이트하고 반환
        new_state = {
            **state,
            "step": 3,
            "sum_result": state["sum_result"] + tens * 10
        }
```

```
        # ✅ interrupt는 상태 업데이트 후에 호출
        interrupt(f"Step 2️⃣ 십의 자리 계산: {a//10} + {b//10} = {tens}")
        return new_state

    # 종료
    return state

from langgraph.graph import END

def route_by_step(state):  # sourcery skip: assign-if-exp, reintroduce-else
    if state["step"] == 1:
        return "ones"
    if state["step"] == 2:
        return "tens"
    return END

# 그래프 구성

graph = StateGraph(AddState)

#graph.add_node("ones", ones_step)
#graph.add_node("tens", tens_step)

# add_node 함수를 ones와 tens 노드 모두에 사용
graph.add_node("ones", add_node)
graph.add_node("tens", add_node)

graph.set_entry_point("ones")

graph.add_conditional_edges(
    "ones",
    route_by_step,
    {
        "ones": "ones",
        "tens": "tens",
        END: END,
    }
)

graph.add_conditional_edges(
    "tens",
    route_by_step,
    {
        "ones": "ones",
        "tens": "tens",
        END: END,
    }
)
```

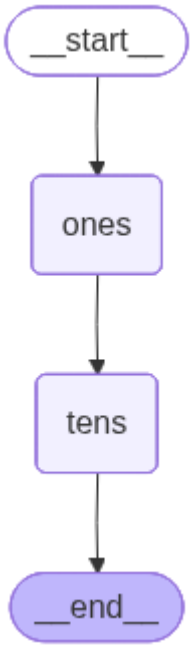
```
checkpointer = MemorySaver()

app = graph.compile(checkpointer=checkpointer)
```

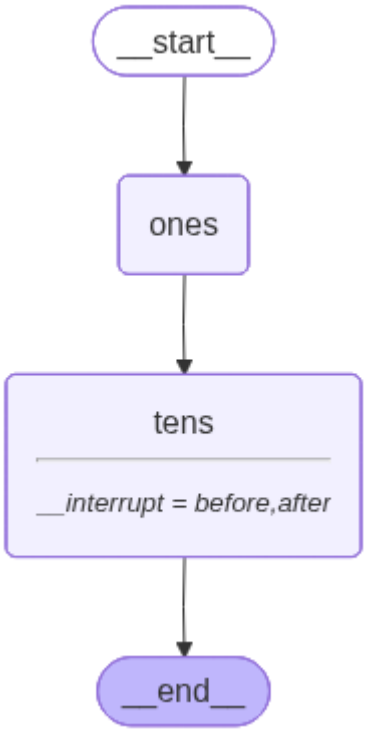
```
# 노드 함수들을 분리

def ones_step(state):
    """일의 자리 계산"""
    a = state["a"]
    b = state["b"]
    ones = (a % 10) + (b % 10)

    return {
        **state,
        "step": 2,
        "sum_result": ones
    }

def tens_step(state):
    """십의 자리 계산"""
    a = state["a"]
    b = state["b"]
    tens = (a // 10) + (b // 10)

    return {
        **state,
        "step": 3,
        "sum_result": state["sum_result"] + tens * 10
    }
```

```
# 그래프 구성
from langgraph.graph import StateGraph, END
from langgraph.checkpoint.memory import MemorySaver

graph = StateGraph(AddState)

# 노드 추가
graph.add_node("ones", ones_step)
graph.add_node("tens", tens_step)

# 엣지 설정
graph.set_entry_point("ones")
graph.add_edge("ones", "tens")
graph.add_edge("tens", END)

# checkpointer와 interrupt 설정
checkpointer = MemorySaver()

# ✅ tens 노드 실행 "전"에 중단 (ones 결과 확인용)
# ✅ END 전에 중단 (tens 결과 확인용)
app = graph.compile(
    checkpointer=checkpointer,
    interrupt_before=["tens"],  # ones 실행 후 멈춤
```

```
    interrupt_after=["tens"]    # tens 실행 후 멈춤
)
```

```
from IPython.display import Image, display

# 그래프를 머메이드 형식의 PNG로 렌더링하여 표시
display(Image(app.get_graph().draw_mermaid_png()))
```

- 셀 출력 (`2.1s`)
  - 

  - 수정된 머메이드 그래프
  - 


```
#  머메이드 텍스트 코드 자체를 확인하고 싶을 때

print(app.get_graph().draw_mermaid())
```

- 위의 다이어그램 텍스트 코드
    - 첫번째

```
---
config:
flowchart:
    curve: linear
---
graph TD;
    __start__([<p>__start__</p>]):::first
    ones(ones)
    tens(tens)
    __end__([<p>__end__</p>]):::last
    __start__ --> ones;
    ones --> tens;
    tens --> __end__;
    classDef default fill:#f2f0ff,line-height:1.2
    classDef first fill-opacity:0
    classDef last fill:#bfb6fc
```

- 두번째

```
---
config:
flowchart:
    curve: linear
---
graph TD;
    __start__([<p>__start__</p>]):::first
    ones(ones)
    tens(tens<hr/><small><em>__interrupt = before,after</em></small>)
    __end__([<p>__end__</p>]):::last
    __start__ --> ones;
    ones --> tens;
    tens --> __end__;
    classDef default fill:#f2f0ff,line-height:1.2
    classDef first fill-opacity:0
    classDef last fill:#bfb6fc
```

---

3) 1단계 승인 후 마지막까지 끝내보기

```
# 디버깅

thread_id = "add-demo-9"  # 새 thread_id

# stream()으로 실행하여 중단점 확인
config = {
    "configurable": {
        "thread_id": thread_id
    }
}

# Step 1 실행 (ones 노드 실행 후 tens 전에 멈춤)
for event in app.stream(
    {
        "a": 12,
        "b": 35,
        "step": 1,
        "sum_result": 0
    },
    config=config
):
    print("📌 Event:", event)

# 현재 상태 확인
current_state = app.get_state(config)
print("\n📌 Current State:", current_state.values)
print("📌 Next Node:", current_state.next)  # 다음에 실행될 노드
```

- 📌 Event: {'ones': {'a': 12, 'b': 35, 'step': 2, 'sum_result': 7}}
- 📌 Event: {'**interrupt**': ()}
- 📌 Current State: {'a': 12, 'b': 35, 'step': 2, 'sum_result': 7}
- 📌 Next Node: ('tens',)

```
# 최초 실행하기
thread_id = "add-demo-10"

config = {
    "configurable": {
        "thread_id": thread_id
    }
}

# ones 실행 후 tens 전에 멈춤
paused_state = app.invoke(
    {
        "a": 12,
        "b": 35,
        "step": 1,
        "sum_result": 0
    },
    config=config
)
```

```
print("📌 paused_state:", paused_state)

# ✅ 중단점 확인: get_state()로 다음 노드 확인
state_snapshot = app.get_state(config)
print("📌 Next node to execute:", state_snapshot.next)  # ('tens',) 출력됨
print("📌 Is interrupted?:", len(state_snapshot.next) > 0)

# Step 1 승인 (None을 전달하면 다음 노드 실행)
print("\n👉 Step 1 승인 - tens 노드 실행...")
continued_state = app.invoke(None, config=config)

print("📌 continued_state:", continued_state)

# 다시 상태 확인
state_snapshot2 = app.get_state(config)
print("📌 Next node:", state_snapshot2.next)  # () 또는 END

# Step 2 승인 (최종 완료)
if state_snapshot2.next:
    print("\n👉 Step 2 승인 - 최종 완료...")
    final_state = app.invoke(None, config=config)
    print("📌 final_state:", final_state)
    print("✅ 최종 결과:", final_state["sum_result"])
```

- 📌 paused_state: {'a': 12, 'b': 35, 'step': 2, 'sum_result': 7}
- 📌 Next node to execute: ('tens',)
- 📌 Is interrupted?: True
- 👉 Step 1 승인 - tens 노드 실행...
- 📌 continued_state: {'a': 12, 'b': 35, 'step': 3, 'sum_result': 47}
- 📌 Next node: ()

- 상황 정리
  - `ones` 노드 실행 → `Step 2`, `sum_result: 7` = **`일의 자리 계산 완료`**
  - *문제점:* *`interrupt_before=["tens"]`* *설정 →* *`tens`* *실행 전에 중단*
    - *`__interrupt__`* *이벤트 발생*

```
# 완전한 HITL 실행 코드

thread_id = "add-demo-12"

config = {
    "configurable": {
        "thread_id": thread_id
    }
}

print("🚀 덧셈 계산 시작: 12 + 35\n")
```

```
# Step 1: ones 노드 실행
print("=" * 50)
print("Step 1️⃣ 일의 자리 계산 중...")
print("=" * 50)

user_approved = False

for event in app.stream(
    {
        "a": 12,
        "b": 35,
        "step": 1,
        "sum_result": 0
    },
    config=config
):
    if "ones" in event:
        result = event["ones"]
        print(f"✅ 일의 자리 계산 완료: {result['a']%10} + {result['b']%10} = {resu

    if "__interrupt__" in event:
        print("\n🛑 중단점 도달!")
        state = app.get_state(config)
        print(f"📊 현재 상태: {state.values}")
        print(f"📍 다음 노드: {state.next}")

        # 사용자 승인 받기
        user_input = input("\n👉 계속 진행하시겠습니까? (y/n): ")

        if user_input.lower() == 'y':
            user_approved = True
        else:
            print("❌ 사용자가 거절했습니다. 중단합니다.")

        break  # ✅ 중요! interrupt 후 루프 탈출

# Step 2: tens 노드 실행
if user_approved:
    print("\n" + "=" * 50)
    print("Step 2️⃣ 십의 자리 계산 중...")
    print("=" * 50)

    user_approved2 = False

    for event in app.stream(None, config=config):
        if "tens" in event:
            result = event["tens"]
            tens_value = (result['sum_result'] - 7) // 10
            print(f"✅ 십의 자리 계산 완료: {result['a']//10} + {result['b']//10} =

        if "__interrupt__" in event:
            print("\n🛑 중단점 도달!")
            state = app.get_state(config)
            print(f"📊 현재 상태: {state.values}")

            # 최종 승인
```

```
            user_input2 = input("\n👉 최종 결과를 확정하시겠습니까? (y/n): ")

            if user_input2.lower() == 'y':
                user_approved2 = True
            else:
                print("❌ 사용자가 거절했습니다.")

            break  # ✅ 중요! interrupt 후 루프 탈출

    # 최종 완료
    if user_approved2:
        final_state = app.invoke(None, config=config)
        print("\n" + "=" * 50)
        print(f"✅ 최종 결과: {final_state['sum_result']}")
        print("=" * 50)
```

- 셀 출력 (`17.5`)

```
🚀 덧셈 계산 시작: 12 + 35

==================================================
Step 1️⃣ 일의 자리 계산 중...
==================================================
✅ 일의 자리 계산 완료: 2 + 5 = 7

🛑 중단점 도달!
📊 현재 상태: {'a': 12, 'b': 35, 'step': 2, 'sum_result': 7}
📍 다음 노드: ('tens',)

==================================================
Step 2️⃣ 십의 자리 계산 중...
==================================================
✅ 십의 자리 계산 완료: 1 + 3 = 4

🛑 중단점 도달!
📊 현재 상태: {'a': 12, 'b': 35, 'step': 3, 'sum_result': 47}

==================================================
✅ 최종 결과: 47
==================================================
```

- next: ***`06. Agentic RAG`***
